DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

VOICE COMMUNICATIONS
OVER PACKET RADIO NETWORKS

by

Seah, Moon Ming

March 1985

Thesis Advisor: J. M. Wozencraft

Approved for public release; distribution is unlimited

T223132

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Voice Communications Over Packet Radio Networks		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; March 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Seah, Moon Ming		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE March 1985
		13. NUMBER OF PAGES 164
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Packet virtual circuit; Voice communications; Routing; Code division multiple access; Network modeling; Link distance

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The use of packet virtual circuit technique for voice communications in military radio networks was investigated. The work was concerned with various aspects of networking which include network modeling, communications techniques, traffic analysis and network control.

An attempt has been made to develop a simple yet efficient time slot assignment algorithm. Performance of this algorithm was

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

analyzed under a variety of slot depths and network topologies using computer simulation. The Erlang' B results were used to provide more insight into the channel characteristics of the packet radio networks. The capabilities of implementing TDMA/CDMA hybrid schemes in the system were scrutinized.

A method to estimate the transmission capacity of the inter-node links was found. We demonstrated its effectiveness in controlling local congestion by computer simulation. Graphical results were presented to highlight the behavior of the proposed packet radio networks. We concluded that an appropriate link weight function would provide efficient and reliable network services.

Approved for public release; distribution is unlimited.

Voice Communications over Packet Radio Networks

by

Seah, Moon Ming
B.Eng(Hons), National University of Singapore, 1980/81

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1985

ABSTRACT

The use of packet virtual circuit technique for voice communications in military radio networks was investigated. The work was concerned with various aspects of networking which include network modeling, communications techniques, traffic analysis and network control.

An attempt has been made to develop a simple yet efficient time slot assignment algorithm. Performance of this algorithm was analyzed under a variety of slot depths and network topologies using computer simulation. The Erlang' B results were used to provide more insight into the channel characteristics of the packet radio networks. The capabilities of implementing TDMA/CDMA hybrid schemes in the system were scrutinized.

A method to estimate the transmission capacity of the inter-node links was found. We demonstrated its effectiveness in controlling local congestion by computer simulation. Graphical results were presented to highlight the behavior of the proposed packet radio networks. We concluded that an appropriate link weight function would provide efficient and reliable network services.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my thesis advisor, Prof. J. M. Wozencraft, for suggesting the basis of this thesis work, and for his invaluable advice and guidance during the course of the thesis.

I would also like to express my appreciation to Prof. R. D. Strum for his support during my study at NPS, to my second reader Prof J. F. Chang and to all others who have helped me directly or indirectly to successfuly complete this thesis.

I. INTRODUCTION

A. BACKGROUND

Modern military communications systems are increasingly adopting the digital method of transmitting voice and data. The reasons are obvious. Several kinds of technology such as VLSI technology, microprocessor and associated memory technology, time division switching technology, digital transmission, voice digitization technique, surface acoustic wave technology (SAW) and new software technology all strongly related to digital communications systems have advanced remarkably. This has resulted in significant reductions in data processing costs, and in particular communications support. From the operational point of view, digital communications offers better signal quality at the expense of larger bandwidth over analog communications by reproducing accurately a sequence of electronic pulses (or "bits") at the receiver. When digital systems are operated in multiple-hop networks, signal regeneration and signal processing do not cause an undue amount of degradation in the signal quality, whereas when analog systems are operated in the same networks, noise accumulates. Another advantage of digital communications systems is that they generally offer higher carrier to interference ratio than analog communications systems do under the same operating conditions.

Digital radio networks or packet radio networks become a natural choice to provide efficient communications among a large number of mobile users in a military tactical environment. In addition, the RF (radio frequency) waveform used by packet radios could provide resistance to jamming,

low probability of intercept (LPI) and low probability of recognition (LPR) for secure tactical use. This subject will be treated in detail in a later chapter.

In a packet radio network, all users are assumed to share a common broadcasting radio frequency and access to the network is controlled by microprocessors in the radios. The use of computer control for channel access can lead to a high throughput and low delay means of interconnection for the community of users. It also allows multiple users to simultaneously access a channel without causing much degradation in the performance of any individual user. Note that the basic operation of the network is transparent to the user. The user only inputs the message (i.e. voice or data) to be delivered with the necessary addressing information, and the network handles all other aspects of routing and reliable delivery of message [Ref. 1].

It is worth mentioning that the current communications requirements in military applications are predominately for voice. This is especially true for military tactical or field operations. We shall therefore focus our study on how to provide efficient and reliable voice communications over packet radio networks.

B. SCOPE OF RESEARCH

Packet radio networks provide multi-access services. A signal generated by a transmitter is received over a wide area by a number of receivers; several transmitters operated in a network may transmit signals simultaneously on a common carrier frequency. If two signals at the same broadcasting frequency overlap in time at a receiver, we could use CDMA (code division multiple access), a spread spectrum technique, to separate them and receive them correctly.

However, if two neighbors transmit at the same time to each other, destructive interference occurs; and in this case we assume that neither will be received correctly. A means for controlling this is for each neighboring radio to transmit in a different time slot. This suggests the use of a TDMA (time division multiple access) slot assignment scheme for multiplexing in a packet radio environment.

The main objectives of this thesis are:

1. to discuss the use of packet virtual circuit (PVC) techniques for voice communications in radio networks.

2. to present a comprehensive study of the operating capabilities of a TDMA/CDMA hybrid system and its operating conditions in the packet radio network.

3. to propose and evaluate a simple but efficient time slot assignment algorithm for the packet virtual circuits. The performance of this proposed algorithm is compared with that of an algorithm used by Tritchler in 1983 by computer simulation in a simple network.

4. to understand the main differences between wire line networks and packet radio networks, and to investigate how to use Erlang's B formula in characterizing voice traffic in the packet radio network.

5. to develop a method to measure the availability and transmission capacity of the inter-node links in the packet radio network. These results allow the YEN routing algorithm to produce desired path assignments aiming at regulating and optimizing traffic flow in the network as a whole.

6. to examine the impact of various path updating periods with this flow control mechanism on the network performance (in terms of the desired grade of service). These activities are simulated with a richly connected radio network in the SIMSCRIPT language for execution on the IBM 3033 system.

II. NETWORK MODELING

A. CLASSIFICATION OF NETWORKS

Communication networks can be classified according to the type of function or service they offer [Ref. 2]. Networks that provide communications among all users are called switched communication networks. Examples of switched communications networks are airline message-exchange networks, Telephone networks and Telex networks. The nonswitched networks provide communication between the user and the network only. Timesharing networks , satellite and television broadcasting networks are examples of this category. From these examples, it is obvious that packet radio networks fall into the category of switched communications networks. Three major types of switched communication networks can be readily found in the literature. We will briefly describe each of these networks here. They are :

1. circuit switched networks that provide fixed connections through the networks between two users for the duration of information exchange.
2. packet switched networks that allow the transfer of information between two users through the routing of packets in the networks. Packets are usually processed and switched in a store and forward manner according to the FIFO (first in first out) discipline.
3. message switched networks that receive and store the entire message in secondary storage at each node, and then transmit it to the neighboring node when the output link becomes available.

In this thesis, we study a circuit-switching like network for voice communications. That is, voice is digitized and packetized before transmission. Data flow in the networks is in packetized form, but virtual connections carry the voice traffic. This technique is known as packet virtual circuit [Refs. 1,3]. It is used to set up a fixed connection through the network for the duration of a voice conversation. These connections are explicitly disestablished by a cancel packet when the called party or the calling party hangs up. Note that the overhead for establishment and disestablishment of the connections are negligible as compared with the durations of voice sessions.

Through the use of a time slot assignment scheme, as presented in the next chapter. this approach offers better utilization of the channel as compared with pure circuit switching. Since voice conversation is real time, voice packets must be received in the order in which they are transmitted and with uniform and constant end-to-end delay. Nonuniform end-to-end delays of a fraction of a second in voice traffic become noticeable and are not desireable. During peak activity periods, pure packet switching using a store and forward process causes excessive nonuniform end-to-end delays for the voice traffic and affects voice intelligibly. Pure packet switching is therefore not suitable for voice communications. Thus, we consider packet virtual circuit to be the best choice for voice communications over radio networks.

B. NETWORK TOPOLOGY

All communications networks may be viewed as composed of a set of nodes and links. Figure 2.1 illustrates a simple network consisting of four nodes and links. Nodes are the switching elements for the communications and links connect

pairs of nodes. From the functional point of view, the network provides a useful service to the users, whereas nodes provide necessary functions required by the system, and links provide communications among nodes.

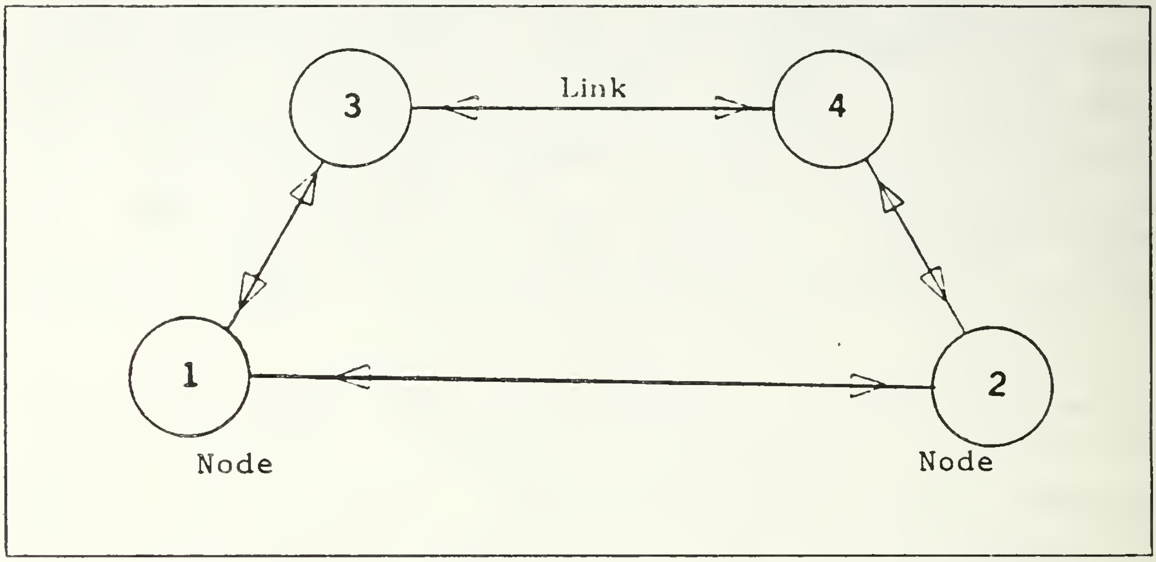


Figure 2.1 A Communications Network

Throughout this thesis, we will make the following assumptions about a node, its neighbors, and a link. A node is a transceiver (receiver-transmitter) with processing capability and a clock. Many functions can be performed simultaneously by parallel processors at each node. All nodes in the network have exactly the same capabilities. Each node may process a given message (packetized voice) differently depending on whether it is a destination node, source node or intermediate node. When one node can pass traffic directly to another node, we say that the nodes are neighbors. Each node in the network is allowed to have a small number of neighbors, and has a transmission range that is small compared to the diameter of the network. Thus

several communications can be supported simultaneously across the network. A link exists between two nodes whenever two way communications is possible between them.

Communication in a radio network is by means of antenna transmissions. If we assume the use of omni-directional antennas for communications, a transmission could then be received by many other nodes besides the addressed node; any node receiving traffic not addressed to it will ignore the message. With the use of code division multiplexing technique, two or more links to a node may exist and operate simultaneously.

As the number of nodes increases, the number of ways one can interconnect the various nodes increased rapidly. That is, many topologies are possible for a network. We will not discuss the efficiency of each of these topologies. Instead, we will just briefly describe a number of network topologies that can be readily found in the literature and their possible military applications. They are :

1. fully connected topology
2. minimal spanning tree topology
3. single-center, single-star topology or centralized topology
4. single-center, multidrop topology
5. multicenter, multistar topology or decentralized topology
6. multicenter, multidrop topology or decentralized topology
7. loop or ring topology

In military applications, strategic and tactical networks may consists of one or more of the above topologies. Tactical networks commonly employ decentralized topologies.

Figure 2.2 illustrates a fully connected network topology. It shows a direct connection between every pair of nodes in the network. The number of links required for such a structure is proportional to the square of N where N is the number of nodes in the network. This topology provides better response time and throughput than any other topologies. Figure 2.3 illustrates a minimal spanning tree topology. This topology uses a minimun number of links to construct a network.

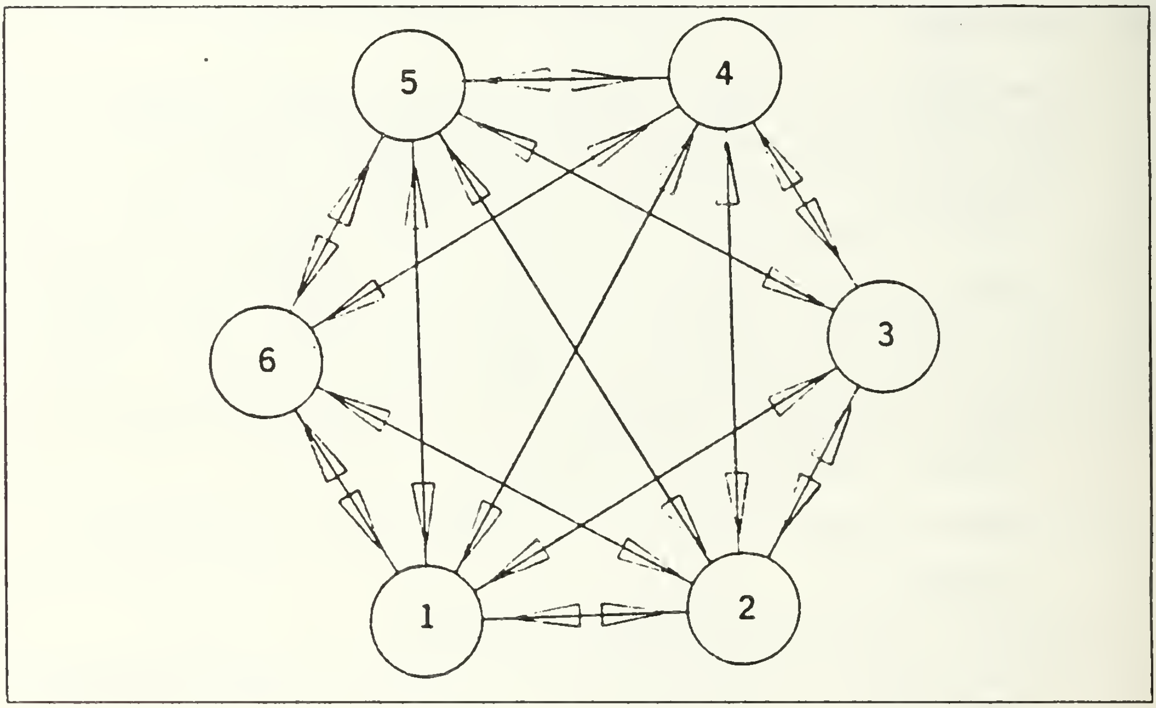


Figure 2.2 A Fully Connected Network

Figure 2.4 illustrates a single-center, single-star topology. This network can be found in military fire control operations. Figure 2.5 shows A single-center, multidrop topology which is a multilevel, hierachical network. Army battalion radio networks use similiar configurations.

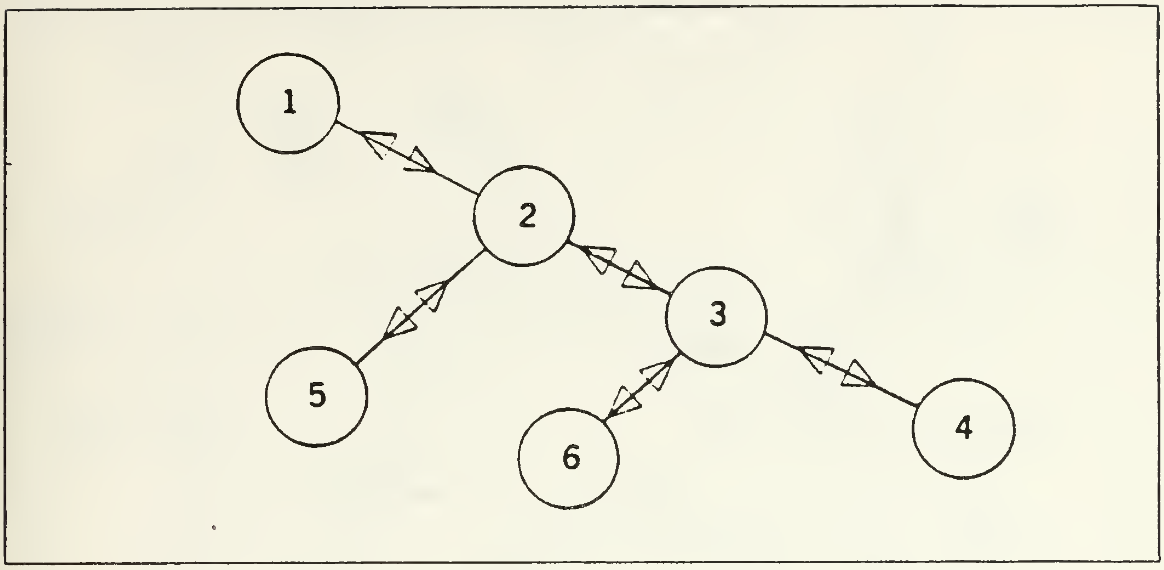


Figure 2.3 A Least-link Network

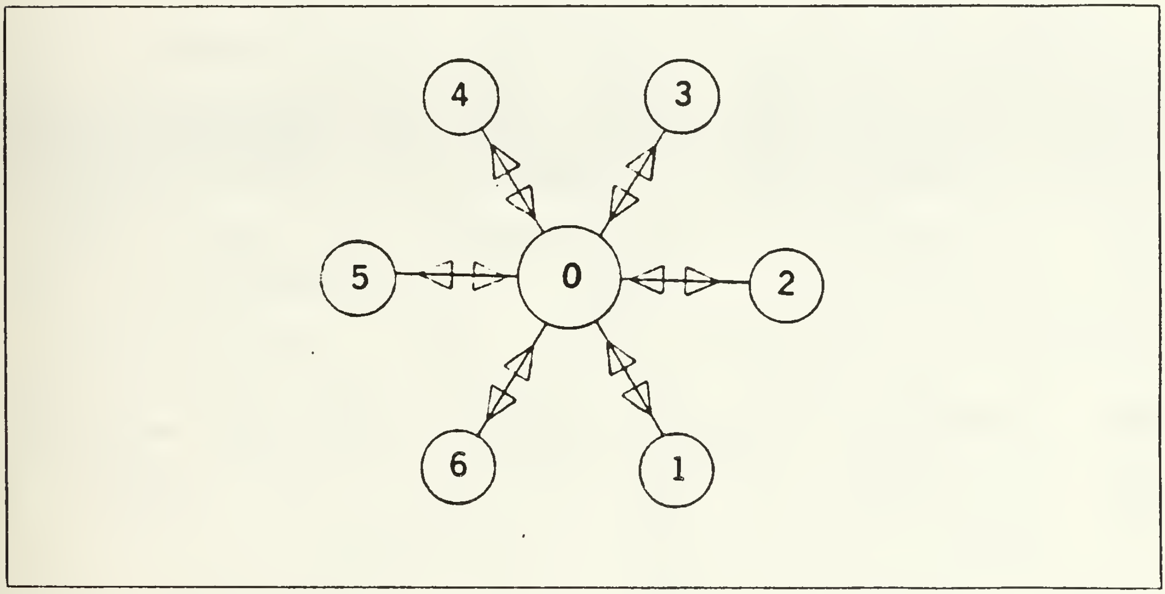


Figure 2.4 A Single-center Single-star Network

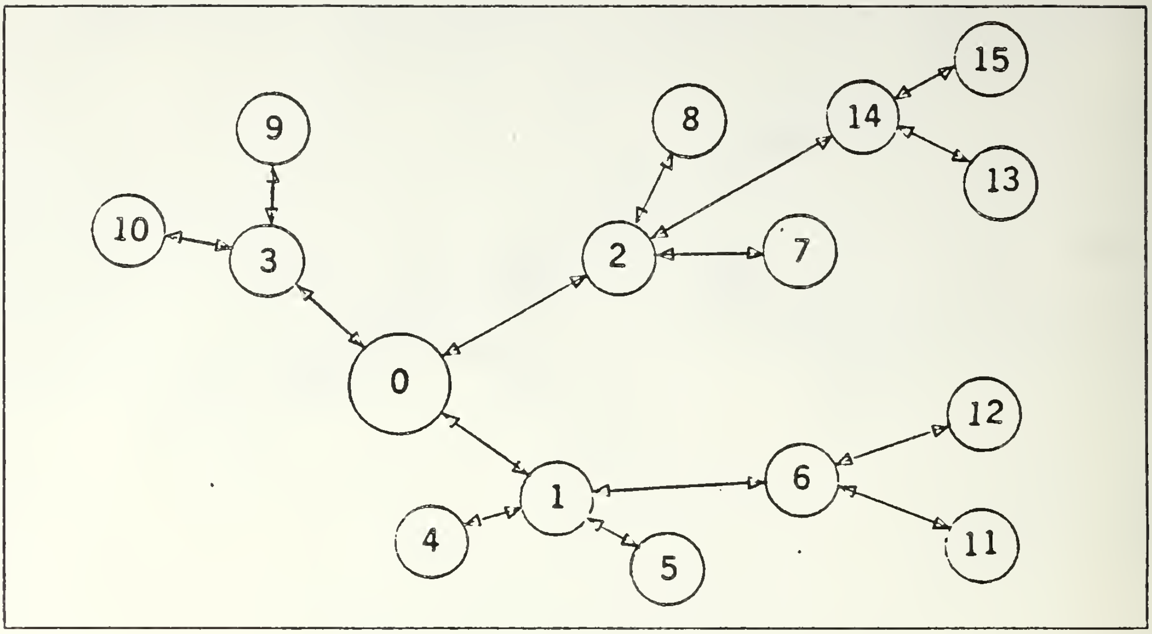


Figure 2.5 A Single-center Multidrop Network

Figure 2.6 and 2.7 illustrate two versions of a decentralized topology. These are widely used in the private, corporate voice and data networks. These topologies are also commonly used in larger military radio networks. Figure 2.8 illustrates a ring topology. This topology poses many difficulties in terms of implementation for bidirectional information flow. Many researchers have worked on this problem. A variety of rings such as token rings, contention rings, slotted rings and register insertion rings are discussed in the literature. [Refs. 4,5,6]

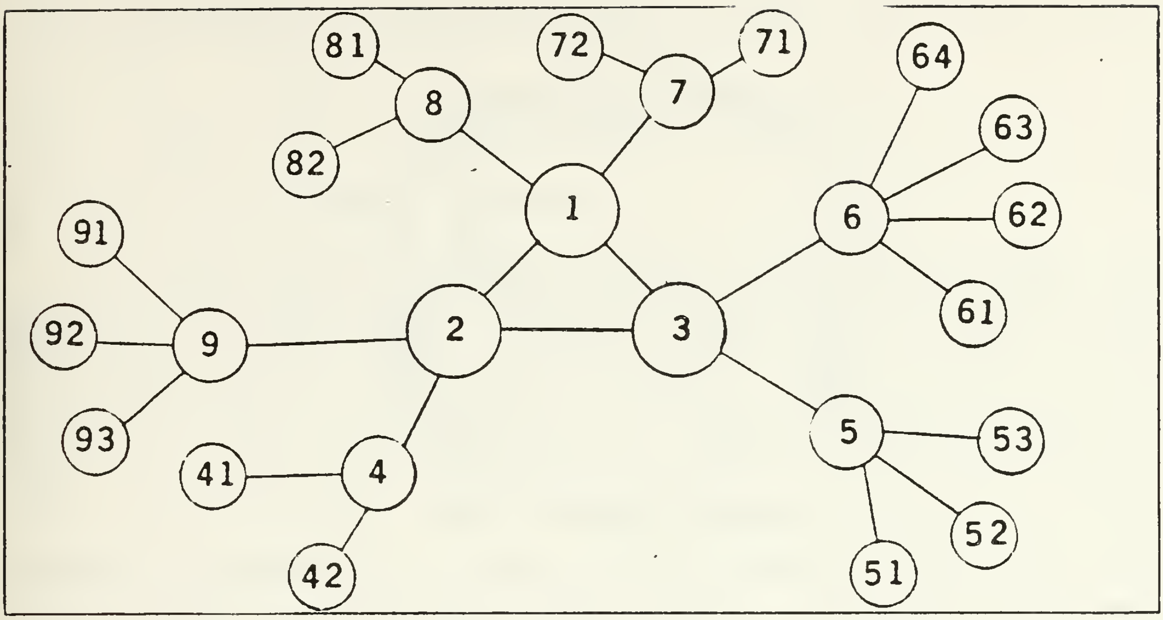


Figure 2.6 A Multicenter Multistar Network

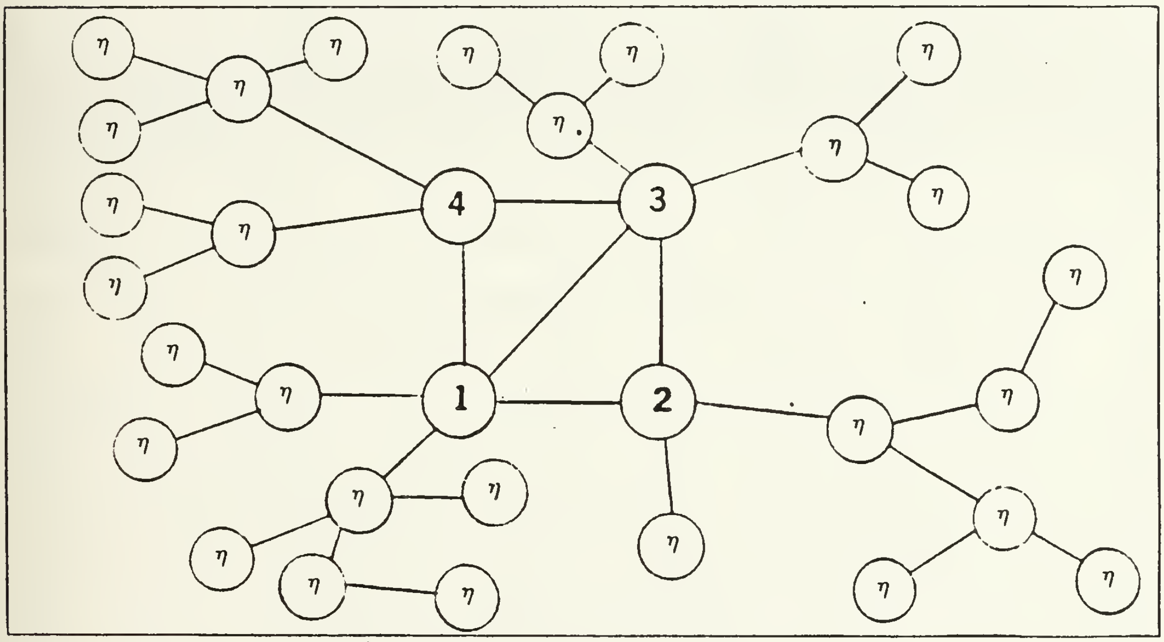


Figure 2.7 A Multicenter Multidrop Network

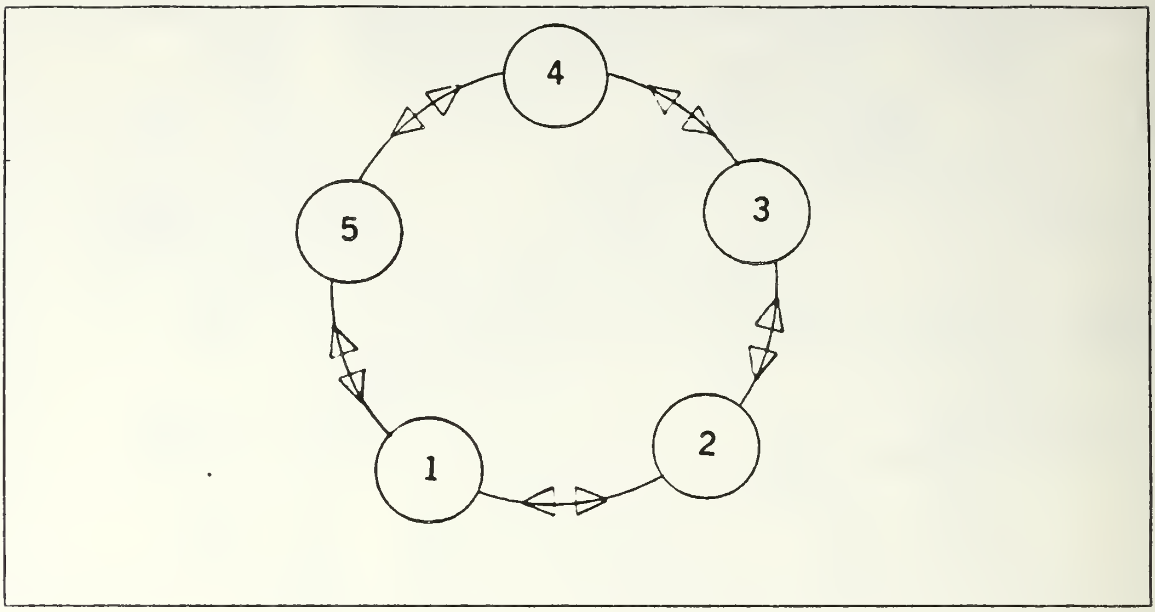


Figure 2.8   A Loop Network

III. COMMUNICATIONS TECHNIQUES

A. MULTIPLE ACCESSING TECHNIQUE

The technique of transmitting and receiving a number of different signals over a single communication path without intefering with each other's signal is referred to as multiple access. By sharing the same communications path with other signals, the cost per bit transmitted may be effectively reduced, even taking account of the increased hardware complexity. The cost effectiveness of the system is based on the average loading of all the communications links. If the overall link utilization is very low, multiple access technique may prove to be inefficient.

There are two widely used multiple access techniques, namely, frequency division multiple access (FDMA) and time division multiple access (TDMA). In FDMA, all users share the frequency spectrum of a communications path and transmit simultaneously. Note that each user is allocated a unique frequency band. If the full bandwidth W of the communications path is divided into N users (or N channels), then each user has the frequency bandwith of W/N. Moreover, each user can transmit only at speeds less than the frequency slot of W/N. The limitation is due to the need of small guard bands between adjacent channels to prevent any sideband signals from overlapping. In TDMA, all users occupy the same RF bandwidth, but transmit sequentially in time. Finally, if we allow users to occupy the same RF bandwidth and also transmit simultaneously over a single communications path, some means of separating the signals at the receiver must be used. Code division multiple access (CDMA) has this capability and is our choice.

In CDMA (also termed direct sequence spread spectrum), the signal's spectrum is spread over a radio frequency channel greater than that necessary to transmit the information. For example, 64 kbps packetized voice may be modulated into a bandwidth of 3.2 MHz rather than 128 KHz. The band spreading is accomplished by means of a PN (pseudo-random) code which is independent of the information. The same code is used at the receiver to correlate with the incoming signal and recover the baseband information. More than one correlator could be used at the receiver to receive different signals simultaneously and recover them separately.

References 7, 8, 9 and others cover many other spectrum techniques such as time hopping and frequency hopping, these are not relevant to our study, and we shall not discuss them.

B. TDMA/CDMA HYBRID SCHEME

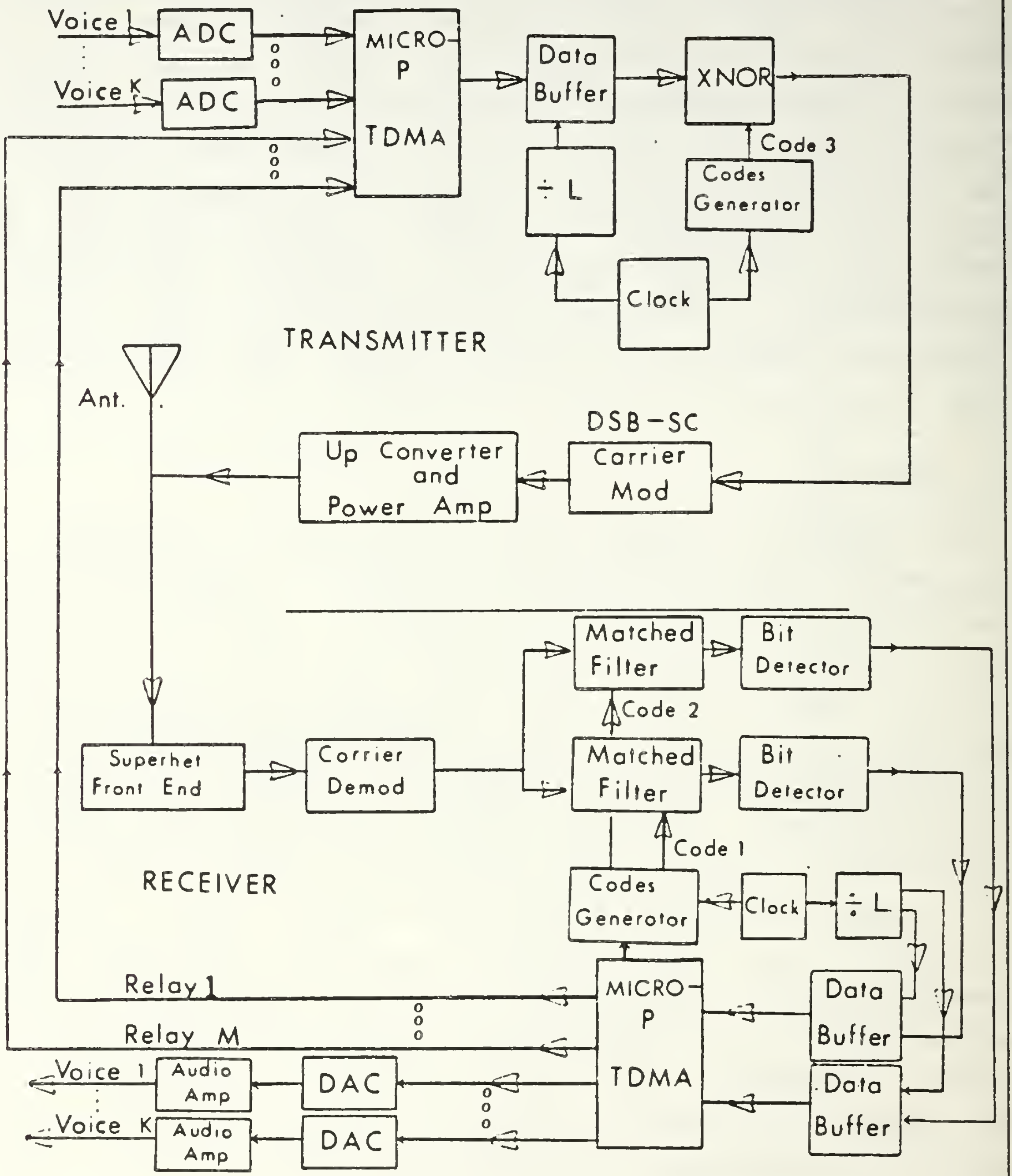
The main advantage of TDMA relative to FDMA is the number of carrier modulation units involved. In TDMA, only one carrier modulation unit is needed, whereas in FDMA each channel requires a separate carrier modulation unit, and therefore N such units are needed. The most important aspect of TDMA is that it does not have intermodulation problem and only performs simple processing on an incoming message before relaying to the next node. That is, TDMA does not require any analog to digital conversion in relaying messages [Ref. 10]. The major problem of TDMA is that it is vulnerable to multipath interference. Since CDMA is a spread spectrum technique, we can incorporate CDMA into TDMA to overcome this problem. We should note that this hybrid technique (TDMA/CDMA) requires accurate synchronization of all the nodes in a packet radio network.

In this hybrid scheme, time is divided into fixed duration frames and each frame is further divided into a number of equal duration time slots. For example, in our work a frame consists of 12 slots and has duration of 125 micro-seconds; thus there exist 8000 frames per second and 12 * 8000 slots per second. Each frame is used to carry a voice packet consisting of a number of bits and each of these bits is modulated with a spreading code before transmission. A conceptual implementation of this hybrid scheme in a packet radio is shown on the next page (Figure 3.1).

C. ADDRESSING

As shown in Figure 3.1, each node is assigned a pseudonoise (PN) code as its identity. Each node must therefore know about all its neighbors codes in order to receive their messages correctly. This provides message privacy. That is, all packets transmitted or relayed by a node to its neighbors will be modulated with the transmitting node's own code sequence to produce the desired wideband signals. At the receiver, the received wideband signals are correlated with a replica of the PN code so as to recover the packets correctly. In the case of relay traffic, a new wideband signal will be generated and retransmitted to the neighbor along the best path to the destination. Since a different correlator is dedicated to receiving messages from each neighbor, the number of correlators needed at the receiver of each node must be equal to the total number of its neighbors.

To recover the baseband information accurately, these codes should provide autocorrelations with low sidelobes and with large amplitude spikes of narrow width. Each code should also have low cross correlation with the codes of the



Note : K voices, M relays

K + M = 6

L = Number of chips per bit

Figure 3.1 A TDMA/CDMA Packet Radio

other users so that they do not interfere with each other. R. Gold in 1967 [Ref. 11] proposed an elegant way to generate good codes. The key for generating Gold codes is to find, for a given maximal length sequence , another equal length maximal length sequence (MLS) that, together with the first one, forms a "prefered pair". Note that maximal length sequence could be easily generated by N stages feedback shift registers. A Gold code will then be formed by taking the XOR (modulo-2 sum) of these two sequences. A total number of $2^n + 1$ different Gold codes can be generated from this prefered pair. An interesting property about these Gold codes is that all cross correlations are bounded. If N is sufficiently large, these cross correlations are extremely low relative to the peak amplitudes of each of the autocorrelations. This keeps bit error rate (BER) of the system extremely low if the input signal to noise ratio is sufficiently high. Bit error rate is commonly used to measure the effectiveness of a digital communications system in the presence of noise and interference.

D. INTERFERENCE REJECTION

As alluded to in the previous sections, a TDMA/CDMA packet radio system uses direct sequence spreading signals for communications. Due to the inherent processing gain of the system, the effect of narrow-band interference can be reduced significantly. Processing gain (PG) is used in spread spectrum systems to measure the effectiveness of the system against noise and interference. It is defined as the ratio between the signal to noise or interference before the code correlator (or matched filter) and after the correlator. It is equal to the length of the PN code sequence used for an information bit (or the number of chips in a bit), that is, PG = L, where L is the number of chips

per bit. The correlation of the received signal with the replica of the PN code reduces the level of the narrow-band interference by spreading it across the frequency band occupied by the wideband signal. The interference becomes a very low noise with a relatively flat spectrum. Conversely, the correlation operation collapses the desired signal to the information signal bandwidth. Thus, the effects of interference due to other users and intentional jamming can be suppressed considerably through the use of this communications technique. The allowable jamming to signal (J/S)dB can be determined by

$$(J/S)dB = (PG - (S/N)out)dB$$

where (S/N)out is the required signal to noise ratio to realize a desired BER. For example, if the processing gain is 20 dB and the output signal to noise ratio requires 10 dB for 10^{-3} BER, then (J/S)dB = 20 - 10 = 10 dB. That is, signals can be detected reliably even when the interference is 10 times greater than the input signal power. This is certainly a powerful means to attenuate and reject interference.

The TDMA/CDMA technique is also very effective in overcoming multipath interference or intersymbol interference. Multiple interfering signals due to reflections from geographical features or man-made objects will arrive at the receiver with different time delays and different amplitudes. These signals will be rejected by the matched filter at the receiver due to the cross correlation operation. That is, the correlation between the spreading code and a delayed version of the transmitted signal produces a very small voltage level relative to that of the direct transmitted signal if the delay is more than a chip's duration.

In short, a TDMA/CDMA system provides better utilization of channel, selective addressing capability for multiple users, low probability of interception, anti-interference and especially antijam capability in a hostile electronic warfare environment.

E. A PROPOSED TIME SLOT ASSIGNMENT ALGORITHM

Based on the capabilities of the TDMA/CDMA hybrid scheme, it is reasonable for us to assume that intelligible voice communications could always be effected if the links and particularly the time slots associated with each link are available. We should next recognize that the user requirements for communications is a random time process. To achieve high call carrying capability in a network, we need an efficient time slot assignment scheme to place and establish each call swiftly and correctly. If a call request is not successful due to the unavailability of a time slot between the calling node and the called node, a busy tone will be issued to the user, and his call request is immediately cleared from the system.

We further state the following conditions for operation of a packet radio network using TDMA/CDMA. First of all, a node is not allowed to transmit and receive simultaneously in a slot. It can either transmit or receive at a time. However, because of the CDMA technique involved, nodes can receive signals from more than one neighbor simultaneously. The maximun number of received signals that can be stacked in a slot is defined as slot depth. For 12 slots and 125 micro-seconds per frame, the desirable slot depth (weighting complexity versus performance) will be shown to be 2 in a later chapter. Next, we assume that each node is listening to its neighbors in any slot in which it is not transmitting. A virtual circuit is constructed for each call

consisting of a pair of time slots on each link in the chain connecting the call source and destination. The slots associated with each virtual circuit will be reserved for the duration of the call. Finally, we assume that the connectivity of the network is static in such a manner that all links remain intact for the duration of each conversation and each assignment process.

The desired assignment algorithm should attempt to distribute the transmitted and received signals uniformly over all slots of a frame and maximize the number of successful calls established across the network. A most important aspect is that it should be easy to implement and yet efficient.

With all these constraints and requirements in mind, we propose a random process time slot assignment algorithm with time-out procedure. We will demonstrate the operation of this proposed algorithm by example. A simple example is to consider how a pair of time slots for a calling node and a called node are arranged for a virtual circuit. Figure 3.2 shows time slot assignments per frame for these two nodes prior to a new call attempt. The symbol X-B and R-B represents a slot in which a node is transmitting to B, and a slot in which a node is receiving from node B, respectively. A pair of slots such as X-B and R-B forms a virtual circuit.

The proposed algorithm can be described as follows:

1. When a user at node A intends to converse with another user at node C, he activates a button with node C's address on the packet radio. The packet radio receives this signal and recognizes a requirement for a virtual circuit. It automatically checks its path assignment table and prepares for

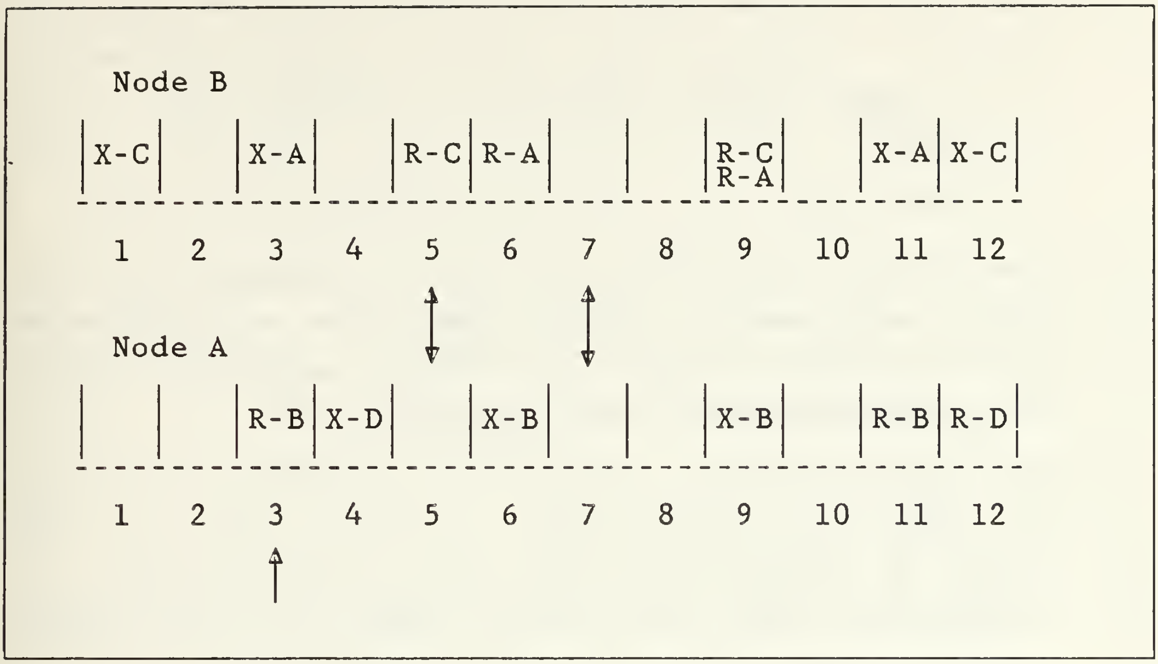


Figure 3.2 Time Slot Assignments for Node A and Node B

handshaking. If its best path neighbor for traffic destined for node C is node B at this moment, it sends RFS (request for service) with a message number and the destination address to node B in the earliest available slot. An available slot for node A to transmit to node B means an empty slot at node A in which node B is not transmitting. If this local call is placed and processed in slot 3, then slot 5 will be the available slot.

2. If the called node (node B) receives the RFS from node A successfully, it responds immediately with a RTR (response to request) with the same message number in an earliest available slot. In this example, slot 7 will be the choice. Note that node B fails to receive the RFS if the total number of received signals in slot 5 has reached the specified slot depth.

3. Once the calling node (node A) receives the RTR from node B, it knows it has made a circuit with node B (i.e. slot 5 and slot 7). It then sends an OK message to node B in slot 5 and waits for an OK message from node C through node B in slot 7.

4. Immediately after receiving the OK message from node C through node B, node A begins to converse with node C. We say that a virtual circuit has been established between node A and node C.

5. If node A does not receive a RTR from node B after one frame time (time-out), it will send the RFS again in whatever available slot closest to slot 5. Node A will make a specified number of such attempts before sending a busy tone to the user.

6. Node B begins to handshake with node C after receiving the OK message from node A. If node B fails to establish a circuit with node C, it sends a BD (breakdown) message via slot 7 to node A. The system in this case will immediately send a busy tone to the user at node A.

IV. TRAFFIC ANALYSIS AND NETWORK CONTROL

A. CONCEPTS OF TRAFFIC

The reliability of a packet radio network depends upon its ability to cope with changes in network topology and traffic. In other words, the packet network must have the ability to react quickly to local disturbances due to statistical peaking of traffic at certain critical nodes in order to provide smooth operation for the whole network. The instantaneous traffic and flow problems can be handled appropriately by certain traffic control techniques, which we discuss below.

Traffic control in telephone networks has been intensively studied, analyzed and used for a long time. Various service disciplines and analytical formulae such as Poisson, Erlang B, Erlang C and others has been developed and are well known in traffic analysis of telephony. The mathematics associated with traffic behavior can be applied equally well to study the characteristics of voice traffic in packet radio networks. To properly apply the result from telephony, it is necessary to understand what is meant by traffic intensities in the proposed packet radio network.

The amount of voice traffic offered to any node in a network is a function of two parameters, the average rate of arrival of new call attempts (λ calls/sec) and the average call duration ($1/\mu$ secs). The product of these two parameters is a measure of traffic intensity ($\rho \triangleq \lambda/\mu$) and is usually expressed in Erlangs in telephony [Ref. 12]. In the packet radio network, we use traffic intensity to indicate the average number of virtual circuits needed (i.e. pairs of

time slots for voice sessions) for each link to serve a given amount of offered traffic (i.e. given λ and μ).

We are now ready to consider the following analysis. We assume each link can carry at most a total of K virtual circuits (or K servers) and that any further new call attempt will be rejected by the network. In all cases, new call attempts will continue to be generated according to a Poisson process. The corresponding state-transition-rate diagram is shown in Figure 4.1 [Ref. 13]

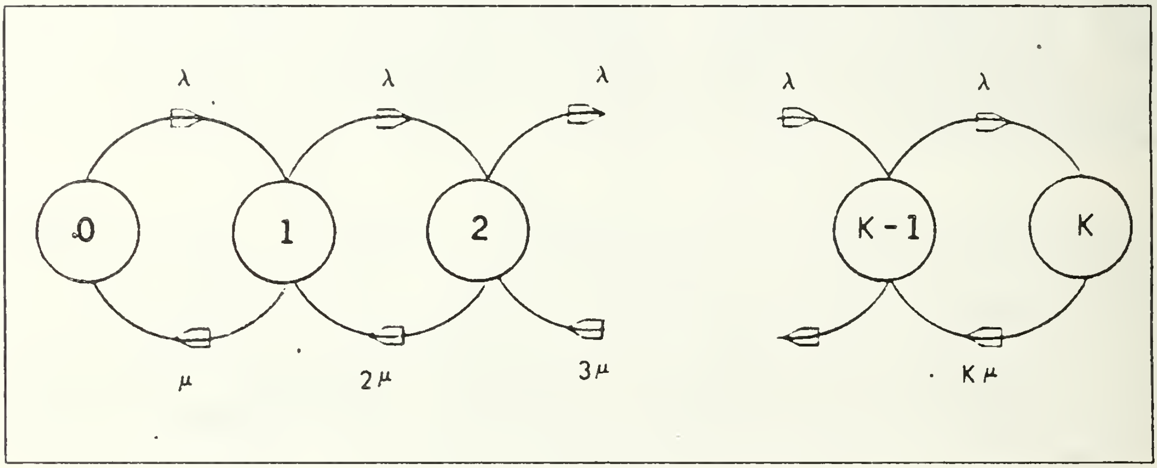


Figure 4.1 State Transition Rate Diagram for K Servers

The coefficients for ith state is given as

$$\lambda_i = \begin{cases} \lambda & , \quad i < K \\ 0 & , \quad i \geq K \end{cases}$$

$$\mu_i = i\mu \qquad i = 1, 2, 3, \ldots, K$$

If P_i denotes the probability that the link has established i circuits at some arbitrary time, it can be shown that

$$P_i = \begin{cases} P_0 \left(\frac{\lambda}{\mu}\right) \left(\frac{1}{i!}\right), & 0 \le i \le K \\ 0, & i > K \end{cases}$$

or

$$P_i = \begin{cases} P_0 \left(\frac{\rho^i}{i!}\right), & 0 \le i \le K \\ 0, & \text{otherwise} \end{cases}$$

where the traffic intensity $\rho \triangleq \lambda / \mu$

Using the conservation relation

$$\sum_0^K P_i = 1,$$

we have

$$P_0 \sum_0^K \frac{\rho^i}{i!} = 1 .$$

Solving for P_0, we have

$$P_0 = \left[\sum_0^K \frac{\rho^i}{i!} \right]$$

P_K can then be determined by

$$P_K = P_0 \frac{\rho^K}{K!}$$

P_K gives the fraction of time that K circuits are used for the link. This expression was first derived by Erlang in 1917 and is usually refered to as the Erlang's B formula. Note that P_K may also be interpreted as the fraction of lost calls.

We can compute the average number of calls ($\bar{N}$) existing in the network as follows.

$$\bar{N} = \sum_{0}^{K} i \; P_i$$

$$= \sum_{1}^{K} i \; P_0 \frac{\rho^i}{i!}$$

$$= P_0 \; \rho \sum_{0}^{K-1} \frac{\rho^i}{i!}$$

Thus far, we have only considered Poisson traffic. Traffic distributions in real situations often deviate from Poisson. A parameter describing different traffic distributions is the variance-to-mean ratio α. α usually varies from 0.5 to 2.0, and Poisson has a unity α. Exact analysis of traffic for non-unity α is relatively difficult. In the literature, various approximation techniques are developed and are available. For simplicity, we shall only use the Poisson distribution in our analysis and simulation work.

From a traffic standpoint, the major difference between telephone networks and packet radio networks concerns the availabilty of time slots over each frame. The telephone network has full frame available for voice traffic, whereas the packet radio network can hardly allow each link to fully utilize all the available time slots in a frame. When a node has more than one neighbor, coordination for communications is not a easy task in a radio network environment. This is simply because a packet radio can not receive and transmit simultaneously, whereas wire line do. Consequently, it is generally impossible for each node to allocate all its time slots with its neighbors, thereby maximizing the number of circuits that each frame can provide.

If we use 12 slots per frame and assume a slot depth of 2 for the radio network, the maximum number of circuits that each link can possibly establish is 8. The average number of circuits we can actually establish for this network would be less than 8, and has been verified experimentally to be about 6. We will discuss this interesting result in a later chapter.

B. NETWORK ROUTING

With a given traffic intensity, the radio network can perform to a expected grade of service (i.e. a specified percentage of lost call) only if routing of the traffic is done in a stable, correct and optimal manner. Routing is defined as a process to find "best paths" for traffic flow in the network. Various criterion functions are possible for "best paths" [Ref. 14]. The criterion for assigning best paths in the packet radio network is to effect maximum traffic throughput (or minimum lost call) in a global manner. This suggests a study of how the entering traffic can be distributed optimally among individual nodes in the network.

Due to the inherently random character of user demands, frequent changes of best paths may be necessary to reflect the new traffic status. The best path of a source-destination node pair is found by computing the shortest distance (or the least cost) over a number of links between them. The distance (or cost) of a link, also refered to as link weight, is just a positive number assigned to each link by the network controller.

Link weight is usually a function of three parameters, the signal to noise characteristic or the attenuation, the processing delay, and the unused transmission capacity of

the link. The relative emphasis on these parameters depends on individual network and user requirements. For example, a network use a fixed weight for all links due to equal transmission capacity of each link, the shortest paths assignment becomes a least hop routing scheme. A hop is defined as a trip over a link. A large variety of other routing schemes concerning the performance of network with these parameters, such as minimum packet delay, least-energy routing and maximum traffic throughput, has been developed in the literature [Refs. 15,16,17]. These routing schemes can be roughly classified as static routing, dynamic routing and hybrid routing. A brief characteristic of each kind of routing method will be presented here.

Static routing, also refered to as non-adaptive routing, makes various assumptions about the node locations and the capacities of the links, and then computes fixed routing tables with these link weights. The path for any node pair is determined prior to the network operation and is independent of normal traffic variations. Although it does not adapt to changes in network traffic, it does provide satisfactory performance on the average over a range of traffic intensities. Least hop routing scheme is a good example. In short, static routing is very useful for networks having constant average traffic statistics. The main advantage of static routing is obviously its simplicity of implementation. The disavantage is that it is not good enough for time-varying traffic situations.

Dynamic routing, also termed adaptive routing, measures and estimates the instantaneous states of the link, and makes routing of messages with these link weights. This scheme ideally routes traffic in the perfect and optimal manner. In practice, it is difficult to measure traffic status instantaneously and accurately. Another practical

constraint is that dynamic routing introduces considerable amounts of overhead traffic for carrying routing updates. Nevertheless, dynamic routing can provide adequate services to the networks having time-varying traffic. A variety of dynamic routing schemes, such as centralized routing and distributed routing, has been implemented in the existing networks. We will briefly describe these two dynamic routing schemes here.

For centralized routing, a central node in the network collects traffic status from all other nodes, processes it and then produces a new set of best paths. One node does all the routing work. All other nodes are not required to perform any routing processing and computation. This scheme thus saves considerably on the amount of hardware needed for the network. However, this scheme is not robust, since a failure of the central node would cause the whole network to become inoperative. Another problem is that routing traffic is concentrated near the central node, and thus affects the throughput efficiency of the network. We can find a number of centralized routing algorithms that were developed from the graph theory in the literature. An algorithm proposed by Dijkstra is [Ref. 18] contained in Apendix A. Because of its simplicity and its computational efficiency, it was used in a distributed manner in our initial simulation work.

For distributed routing, each node constructs its own routing table using periodic updating information from neighboring nodes. No global knowledge of the topology is needed. Each node knows only its neighbors and chooses a prefered neighbor for each destination node. To implement the Dijkstra algorithm in a distributed manner, a flooding technique could be used for each node to inform all other nodes about its current traffic status (or new link weight). After receiving the traffic update information, each node

can then proceed with its own routing computation using the Dijkstra algorithm.

Hybrid routing is considered to be the most attractive scheme for packet radio networks. It uses a link weight function which can produce approximately constant weight when deterministic network conditions dominate and highly varying weights when dynamic network conditions dominate. A proposed link weight function, which can allow the routing to behave in the above mentioned manner, will be presented and discussed in the next section. A routing algorithm proposed by Yen [Ref. 19] which operates in a truly decentralized network, was chosen for investigating the performance and characteristics of this proposed link weight function in the radio network by computer simulation.

Logan in 1983 [Ref. 20] conducted an intensive study on the Yen shortest path algorithm. In his thesis, he mentioned that the Yen algorithm is superior in terms of computational efficiency to many other distributed routing algorithms. It is worth noting that the Yen routing algorithm provides slighty better computational efficiency than the modified Dijkstra algorithm. The attractiveness of the Yen algorithm is that it requires no knowledge of the complete network topology and little information is exchanged between network nodes. The description of the Yen shortest path algorithm is contained in Appendix B.

C. CONGESTION CONTROL

In general, flow control in a communication network serves to optimally allocate resources to satisfy user demands as long as there are resources and to settle contention when the network runs out of resources [Ref. 21]. In the packet radio network, the way to settle the

contention is simple and straight forward. An entering voice call is rejected and lost from the system when the network uses up all its available channels (i.e. traffic limit) and no longer able to accomodate the traffic.

In addition, traffic may get lost due to a local congestion problem because of a poor routing algorithm or inappropriate link weight function. Assuming the Yen distributed routing is perfect in the sense that it can direct and steer traffic correctly according to a given link weight function, a proper distance function must be devised to evaluate current traffic status accurately for each link in the network We believe that an appropriate routing can enhance the network throughput at various traffic intensities before reaching the traffic limit. That is, the routing with a desirable distance function can maintain maximum throughput, retard congestion, and if congestion has to occur, it occurs at maximum throughput.

Based on these requirements, we propose a distance function below considering path attenuation, link congestion level (or transmission capacity) and packet processing time.

$$\begin{aligned} \text{DISTANCE(B,A)} = \text{ATTENUATION(B,A)} &+ \text{P.DELAY(B,A)} \\ &+ \text{C.LEVEL(B,A)} \end{aligned}$$

DISTANCE(B,A) computes link distance (or link weight) from node A to node B. ATTENUATION(B,A) signifies the signal to noise characteristic of the link connecting node A to node B. P.DELAY(B,A) includes processing delays both at node A and node B. C.LEVEL(B,A) indicates indirectly the available number of channels that node A can transmit to node B. The detailed description and analysis of the C.LEVEL is contained in Appendix C.

The distance function will attempt to interrelate all these parameters and produce desired distances. A proper assignment of each of these parameter is crucial to the routing strategy and affects the network throughput. When deterministic network conditions prevail, we want the distance function to produce weights for least-energy routing [Ref. 22]. In the case of equal attenuation for each link, least-energy routing becomes least-hop routing. Least-energy is the path over a link or a series of links with the least attenuation. This routing strategy may pose a serious problem that a relatively small number of critical links will pass the majority of traffic. [Ref. 23] When dynamic network conditions dominate, the distance function should produce highly varying weights which allow the routing to avoid local congestion. Both can be achieved by assigning varying weights for C.LEVEL and unvarying weights for ATTENUATION and P.DELAY.

The C.LEVEL should be inversely proportional to the available number of channels that a link currently has. This inverse function is effective in the sense that when a link is congested, a very high weight will be assigned on that link, and the routing will try to avoid passing traffic through the link. C.LEVEL should therefore be assigned a wider range of possible weights relative to ATTENUATION and P.DELAY.

Simulation results, presented in a later chapter, support to a certain extent the correctness and the effectiveness of the proposed distance function. Note that if minimun packet delay is our main concern, then the P.DELAY will be assigned a higher weight than the other two parameters.

V. SIMULATION

We conduct simulation when a system cannot be studied directly because it does not yet exist, or is not available, or it is too costly to work directly with it. Simulation is also conducted when there are many detailed questions that are very hard to answer through analysis alone.

In general, a simulation study is a way of trying out designs and plans before their actual operations or productions, and also a way of acquiring new knowledge and providing useful insights about a system or an object. Simulations have a wide range of applications such as aircraft evaluation in wind tunnel, war games, flight and space simulation, queuing systems and others. [Ref. 24]

Since the advent of digital computers, simulation has become a practical technique and has made significant contributions to both theory and practice which ranges from the validation of analytical models to the creation of new systems. This is especially true in the fields of control systems, computer systems and communications networks. Complex interactions among interdependent system or units, which are impossible or extremely difficult to study by conventional analytical methods, can be investigated and examined when computers execute simulation programs. The great strength of simulation is that it allows a model of a system or a problem to be developed, tested and analyzed step by step.

It is worth mentioning that simulations require large amounts of detailed knowledge about the structure of the system and about patterns of usage. Simulation is therefore

either limited by its simplifying assumptions, or else be of the same order of complexity as the real system. In addition, it is also important that the simulations be accompanied by analysis that can give order-of-magnitude estimates to ensure that the simulation results are reasonable.

In this thesis, the representation of the system , the rules and relationships that describe it, is defined as the model. The use of the model under specific conditions is defined as simulation. The running of the model on the digital computer is the computer simulation of the system. Our purpose of simulation is to investigate the system behavior and performance. Our approach of simulation is to develop a model and verify that it is a valid one; then a series of parametric simulations can be run to gain understanding of system behavior and performance. To this end, we have used and describe next two networks, one simplified and the other richly connected.

A. A SIMPLIFIED NETWORK

1. Goals of Simulation

A simple model will be used to evaluate the performance of the TDMA/CDMA time slot assignment algorithm we have developed in a previous chapter for a military packet radio network. This model is tested on an IBM 3033 system digital computer under various traffic intensities and slot-depth assignments. Performance comparisons are made between the Tritchler algorithm and the proposed algorithm. An optimum slot depth will be determined from the simulation results. In addition, we employ Erlang's B results (by considering K circuits to be 6) to verify that the model and simulation results are reasonable. This allows

us to proceed with a more sophisticated model later. Another objective of this simulation is to observe and confirm the difference between wire lines and radio links.

2. Description of Model

Small networks (as parts of large networks) of four to eight nodes are connected in the topology as shown in Figure 5.1 Network symmetry is observed such that the numbers of neighbors of node A is equal to that of node B. We use this model to scrutinize the two time slot schemes without involving any routing algorithm.

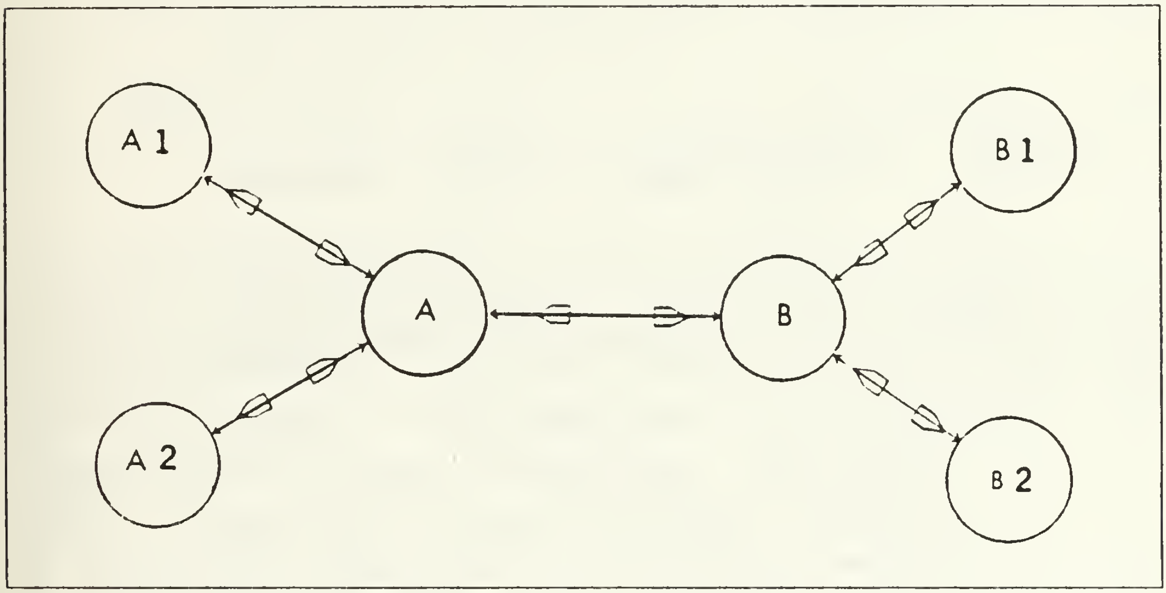


Figure 5.1 A Simplified Network

In this model, a virtual circuit could be a path constructed from node A to node B, or node A via node B to node B1 or node B2, or node B to node A, or node B via node A to node A1 or node A2. That is, only node A and node B are source nodes and the remaining nodes are treated as destination nodes. The following parameters are used in the simulations:

1. Node A1 and node A2 represent an aggregate of nodes connected to node A. The number of these nodes is an input parameter to a simulation run. It is assigned a value from 1 to 3 . The variation of this parameter is to test the robustness and stability of the system under different network topologies.

2. A typical call distribution as a function of time in the modeled network is shown in Figure 5.2

```
|--*----------*------*---------*-------*---->  time
   †          †      †         †       †
From A     From A  From B   From B   From A
  to B       to B    to A     to A     to B
```

Figure 5.2 A Typical Call Distribution

New calls are generated according to an exponential distribution with a mean value ranging from 0.05 secs to 0.35 secs. Calls created from node A and node B are equally probable. Each node has equal chance to be a destination node.

3. The holding time of virtual circuit obeys an exponential distribution function with a mean value of 20 secs.

4. Since human voice has a 4 KHz bandwidth, we can sample at 8 KHz (Nyquist rate). The repetition period of each frame is therefore 1/8000 secs.

Each frame consists of 12 time slots and hence all time slots are 10.417 micro-secs long.

5. The slot-depth is assigned a value from 1 to 4 for each simulation run.

6. Each simulation run is 400 secs long.

We will briefly describe the Tritchler [Ref. 22] time slot algorithm here. His algorithm requires neighbors to exchange coordination messages so as to arrange a pair of time slots for a circuit. Three coordination messages are used. They are :

1. initial request for service from the calling node to the called node

2. response request for service from the called node back to the calling node

3. final assignment notice from the calling node to the called node

In addition, if the called node is not the destination node, a "OK" message or a "BD" message as discussed previously in our proposed algorithm will also be needed for the source node to arrange a virtual circuit via the intermediate nodes to the destination node. During the assignment, each node seeks to conserve its unassigned slots by stacking the received signals whenever possible to a specified slot-depth in a minumum number of slots. A node can receive a coordination message from a neighbor in any slot in which it is not transmitting. Note that his algorithm has assumed that the called node can always receive a coordination message from its calling node as long as the called node is not transmitting. This implies that a slot may have to receive one signal more than its slot-depth.

B. A RICHLY CONNECTED NETWORK

1. Goals of Simulation

A more complete and realistic model is developed to study the real-time operations of the packet radio networks employing the proposed communications and control techniques. Proper operation of the routing and congestion control will be examined in detail.

For simplicity, we will first use the Dijkstra routing algorithm in a distributed manner with the assumption that routing updates are received correctly by all nodes in the network without actually allocating time slots to carry the updating traffic. This model, though relatively simple, is able to measure the performance of the proposed dynamic control technique with different traffic intensities.

However, in order to make the model resemble the object system as closely as possible, we will allocate time slots for carrying routing updates and implement the Yen routing algorithm in the network after gaining enough experience from the previous simulation work. Note that the Yen routing algorithm is believed to be efficient and suitable for decentralized networks in the real world, but it is difficult to be implemented in the simulation model.

2. Description of Model

Figure 5.3 illustrates the network to be used for simulation. It is composed of eleven nodes and twenty-two links. This test network is richly connected and fully decentralized so that we can generate the necessary dynamic network conditions for testing the proposed congestion control mechanism.

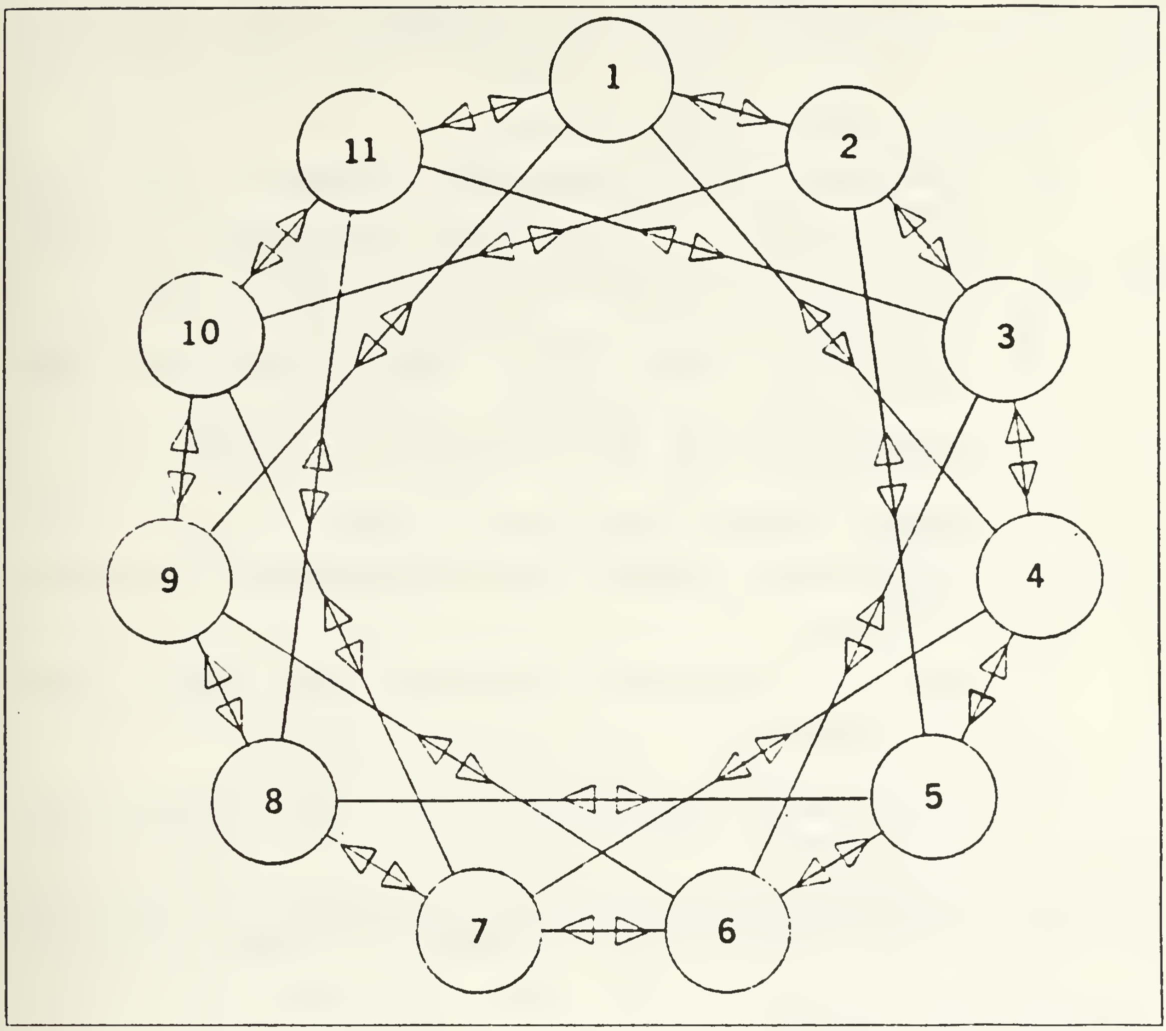


Figure 5.3 A Richly Connected Test Network

To keep the simulation focussed on its goals, we first construct a relatively simple model by employing the Dijkstra routing algorithm and assuming each node in the network has knowledge of all link weights without actually passing the updating information by any time slot. After we program it, run it and draw conclusions, we will proceed to a finer model in which updating traffic is competing with voice traffic for time slot allocation. Instead of the Dijkstra algorithm, we will use the Yen routing algorithm to

reduce the considerable amount of overhead traffic caused by using time slots for routing updates. The parameters used in the simulations are given below.

1. Each link will be assigned a unique ATTENUATION value and a constant P.DELAY value which are the same in either direction for a pair of nodes.

2. A C.LEVEL value will be assigned on each link according to traffic intensities at that particular link at the time of updating.

3. Update periods vary from 2 secs to 15 secs. An updating period will be used for each simulation run.

4. Entering traffic has the same probability distributions and assignments as before.

5. Time frames and slots have the same durations as before.

6. Each simulation run is 800 secs long.

C. SIMULATION LANGUAGE

There are quite a number of high level programming languages available today. Some programming languages are specifically developed for the solution of equations, some are written for simulations of complex organizations and systems, and others are designed for computer assisted instruction.

FORTRAN is one of the most widely used languages for expressing mathematical relationships [Ref. 25]. It has excellent mathematical capability and is therefore fully equippped and supported by our computer center. However, FORTRAN is weak in list processing and logic models. In

particular, it is not a language for simultaneous events or discrete- event simulation.

Due to the discrete nature of packet radio networks, we will use a discrete-event programming language, that is SIMSCRIPT II.5, for our simulation work. SIMSCRIPT is the most widely used simulation language next to GPSS (General Purpose Simulation System). It was developed by RAND Corporation and has been in use since 1962.

SIMSCRIPT is an English-like and free-form simulation language. The statements are understandable directly by someone with a minumum exposure to the language. List processing capabilities are strong. FIFO and LIFO and ordered data structures are easily established. In addition, complex situations can be well structured.

In a SIMSCRIPT simulation, an object system is represented as sets of temporary or permanent entities, each with attributes that have individual values. The basic unit of action for carrying out the simulation is an activity. The object system is modeled and characterized by a number of activites. When operating, these activities reproduce the time-dependent behavior of the system being simulated. When each activity occurs, the system state changes accordingly. The state of the system is changed by either creating or destroying entities, or by changing the attribute values.

In order to simulate the object system accurately, we must therefore model correctly the things that activities do and arrange the execution of subprograms in a proper sequence that represent activities. The order of performing activites within the model corresponds to the order in which the same activites occur in the object system. An instant in time at which an activity starts or stops is called an event. SIMSCRIPT contains a built-in next event timing

routine. This timing routine is the master controller of a simulation run. Simulation terminates when no further events are scheduled [Ref. 26].

D. SIMULATION PROGRAMS

Appendix D contains the simulation program focussing investigation on the proposed time slot assignment algorithm. Appendix E contains the simulation program for the model using the Dijkstra routing algorithm. Finally, Appendix F gives the simulation program for the most complete representation of the proposed packet radio network, using the Yen routing algorithm.

All the simulaion programs are modularly structured. Each simulation program is composed of a preamble, main routine, initialization,events and routines of the simulation model. The preamble has temporary entities, event notices, and miscellaneous declarations. The main routine initializes the model for each new simulaion run and transfers control to the timing routine when initialization is complete. The routine FRESH INPUT (initialization) reads in all the data needed for the simulation and resets all counters. Six events are constructed for handshaking (i.e. time slots assignment), virtual circuit establishment and virtual circuit disestablishment. The event TERMINATION terminates a simulation run. There are also routines for the routing algorithm to perform desired path assignments, distance computation and the collection of statistics.

Statistical phenomena about the operation of a model is controlled by pseudorandom number generators. SIMSCRIPT II provides eleven statistical functions for generating indepedent pseudorandom numbers. Three commonly encountered functions are used in our simulation : Exponential.F,

Randi.F and Uniform.F. Exponential.F is used to generate activities times. Randi.F is used for time slots allocation and Uniform.F is used for generating various link ATTENUATION's.

VI. RESULTS AND DISCUSSIONS

We tackled the analysis of the proposed packet radio networks in a number of stages as described in the previous chapter. We will present the corresponding simulation results in a proper sequence and discuss them accordingly in this chapter.

Performance of the simulation models were measured by two parameters: the average call setup time and the grade of service. Call setup time is defined as the connection time between the moment the last digit is keyed and the moment the virtual circuit between the source and destination nodes is made. Since the simulation process is somewhat random, a practical way to specify this parameter is to define the mean value for the connection times. A lost call is caused by the unavailability of any link connecting the source and destination nodes. We define the grade of service to be the number of successful calls over the total number of calls entering the network.

We also use Erlang's result with appropriate traffic intensities and number of servers to provide a reliable performance evaluation of the simulation model and to provide more insight into the channel characteristic of the packet radio network.

A. RESULTS FOR TIME SLOT ASSIGNMENT SCHEMES

An analysis of the proposed time slot assignment scheme for various possible values of N and X between two nodes is included as Appendix C. Table I contains the analytical result for the probability that the first RFS message

(request for service) fails to travel from a calling node to a called node. The table shows, as expected, that a higher number of calls that can be established as we increase the allowable slot depth. Increasing the slot depth from 1 to 2 generally results in about 5 to 10 percent improvement. There is less improvement by increasing the slot depth from 2 to 3. This fact is further supported by the simulation results shown in Figure 6.1 . It can be seen from this graph that slot depth of 4 only performs slighty better than slot depth of 2. We therefore conclude that slot depth of 2 is sufficient for the proposed packet radio.

The performance of the proposed time slot assignment algorithm was then evaluated for different network topologies varying from 4 nodes to 8 nodes. The corresponding simulation results are given in Figure 6.2, 6.3 and 6.4 . The detailed simulation results are contained in Table II . Both the Tritchler and the proposed algorithm are plotted on the same graphs for easy reference and comparison. Erlang's B results with 6 servers are also shown on all the graphs.

For 4 nodes network, it appears that the network closely resembles a line network having 6 dedicated lines. The Erlang's B results fit quite well to the simulation result of 4 nodes network and therefore validates the simulation model used. We note that each node in the packet radio network has 8 possible channels but can only provide service for 6 channels. This is to be expected since slot matching is a major problem when a node is having more than one neighbor in the packet radio network. Radio network thus performs somewhat worse than wire line network based on same amount of channels under same traffic loads.

The performance results for 6 nodes and 8 nodes networks are very close to each other. This indicates that the proposed time slot algorithm is robust and stable with respect to the changes in network topology.

In most cases, the proposed time slot algorithm performs better than the Tritchler algorithm. This may be due to the simple time-out procedure that the proposed time slot algorithm adopts. Moreover, the proposed time slot scheme may give more uniform usage of time slots in each time frame. In addition, it requires less mean call setup time than the Tritchler algorithm.

TABLE I

A RFS Message from a Calling Node to a Called Node

N_A	X_B	Probability of failure of first attempt		
		S = 1	S = 2	S = 3
3	1	0.090	0.090	0.090
3	2	0.200	0.150	0.150
3	3	0.333	0.278	0.222
3	4	0.500	0.375	0.375
3	5	0.708	0.569	0.498
3	6	0.812	0.685	0.652
4	1	0.090	0.090	0.090
4	2	0.200	0.150	0.150
4	3	0.333	0.278	0.222
4	4	0.500	0.375	0.375
4	5	0.692	0.562	0.493
4	6	0.781	0.657	0.630

Table I

A RFS Message from a Calling Node to a Called Node (Cont'd.)

5	1	0.090	0.090	0.090
5	2	0.200	0.150	0.150
5	3	0.333	0.278	0.222
5	4	0.498	0.374	0.374
5	5	0.662	0.547	0.482
5	6	0.742	0.616	0.596
6	1	0.090	0.090	0.090
6	2	0.200	0.150	0.150
6	3	0.333	0.278	0.222
6	4	0.490	0.371	0.370
6	5	0.618	0.515	0.456
6	6	0.696	0.560	0.546
7	1	0.090	0.090	0.090
7	2	0.200	0.150	0.150
7	3	0.332	0.277	0.222
7	4	0.467	0.357	0.357
7	5	0.563	0.464	0.409
8	1	0.090	0.090	0.090
8	2	0.200	0.150	0.150
8	3	0.323	0.271	0.218
8	4	0.422	0.321	0.321
9	1	0.090	0.090	0.090
9	2	0.197	0.149	0.149
9	3	0.293	0.245	0.197

Note : S = slot depth

N_A = number of empty slots at node A in which node B is not transmitting

X_B = number of slots in which node B is transmitting

Prob. of failure = $\bar{Y} / N_A$

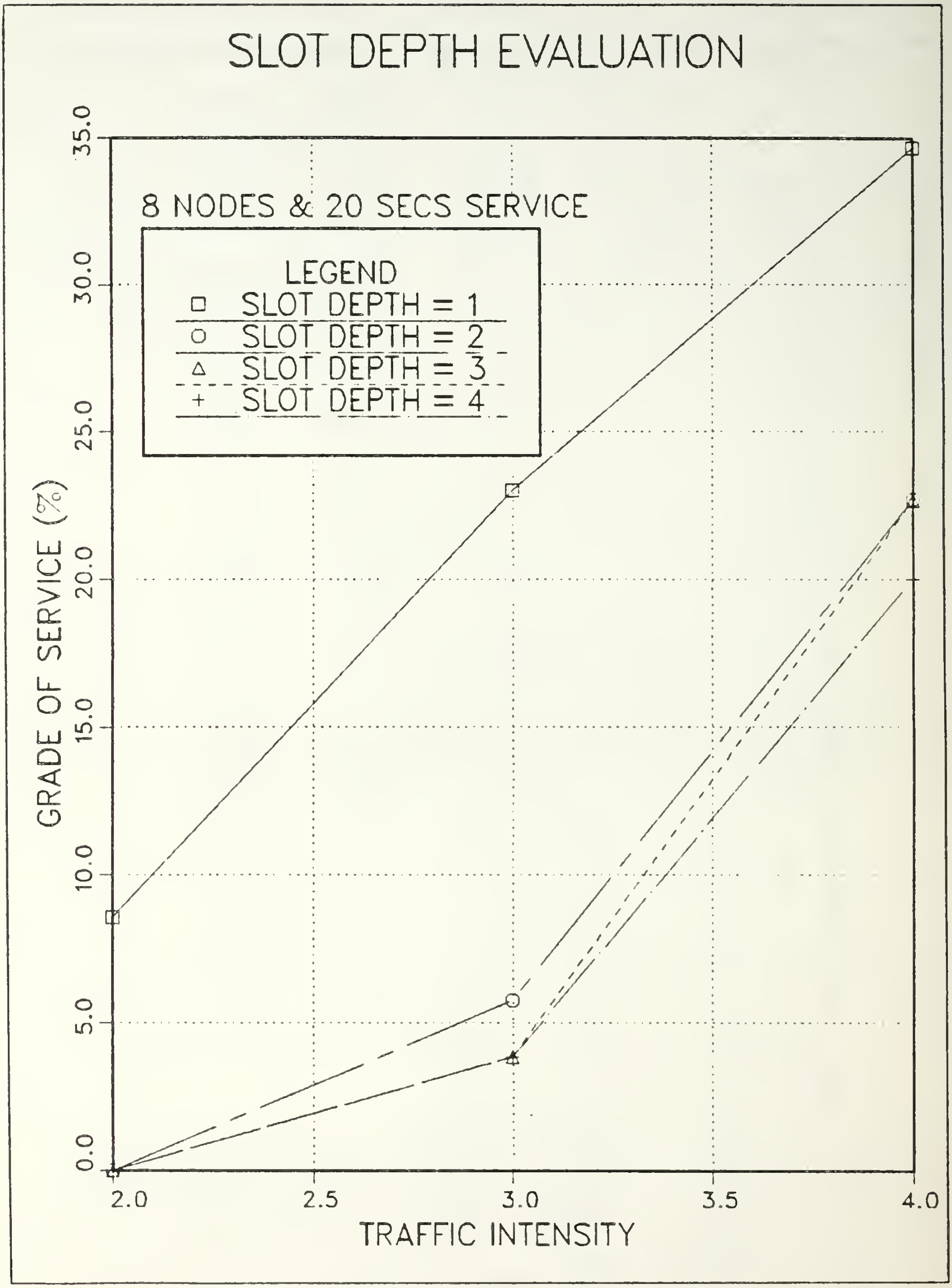


Figure 6.1 Results for Slot Depth Comparison

TABLE II

Results for the Simplified Network

Traffic intensity	Nodes	Percentage of lost call		Mean call setup time (micro-secs)	
		Tritchler	Proposed	Tritcher	Proposed
2	4	2.86	0.00	174	140
	6	2.86	2.86	192	169
	8	2.86	2.86	194	169
3	4	3.85	0.00	177	164
	6	3.85	3.85	205	192
	8	3.85	5.77	212	176
4	4	16.00	16.00	224	168
	6	22.67	20.00	225	201
	8	22.67	22.67	230	214
5	4	21.35	21.37	220	174
	6	26.79	25.85	244	192
	8	27.68	23.44	220	207
6	4	29.73	28.83	253	202
	6	35.14	33.33	268	209
	8	36.04	33.33	246	217

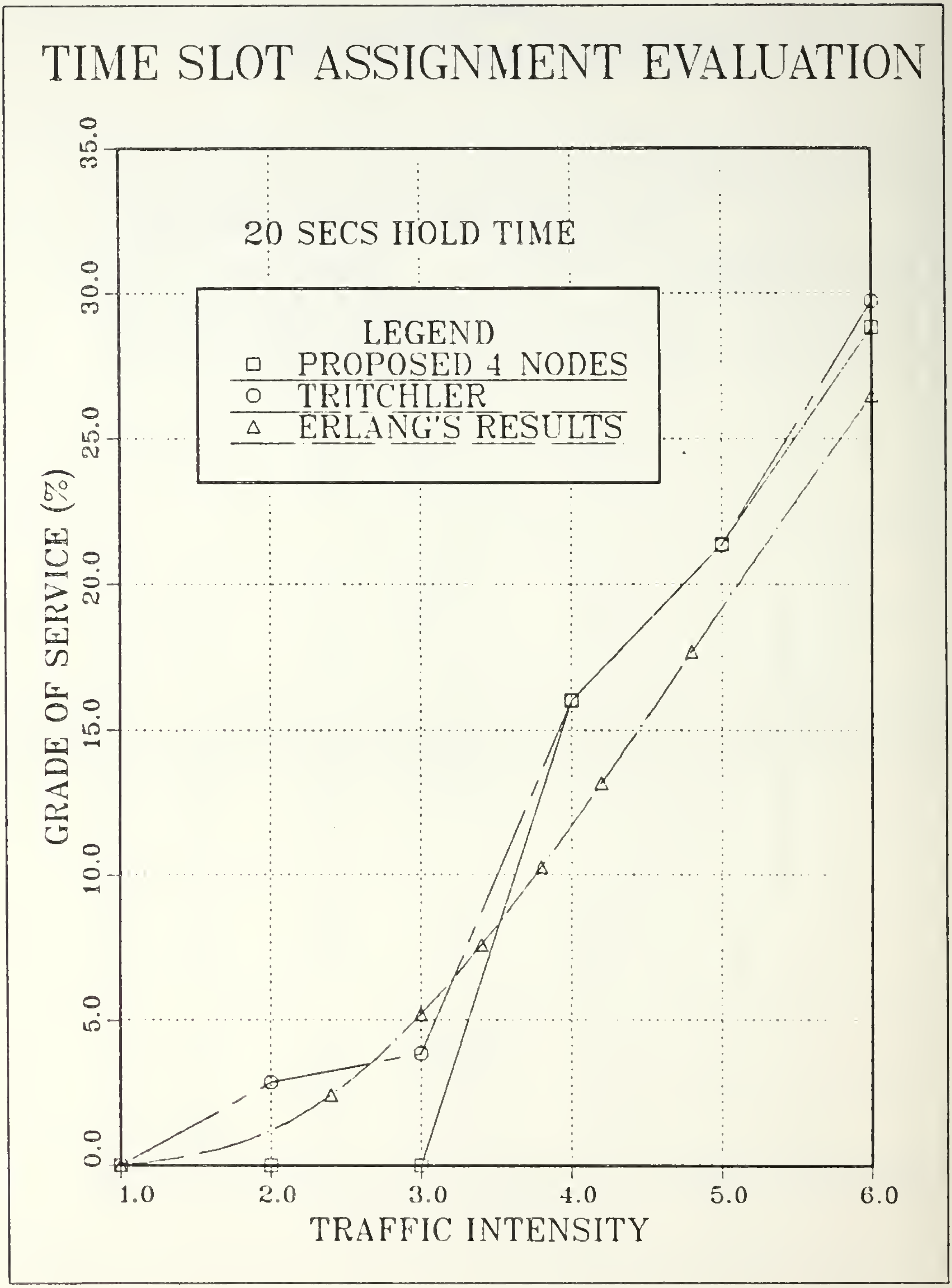


Figure 6.2 Results for 4 Nodes Network

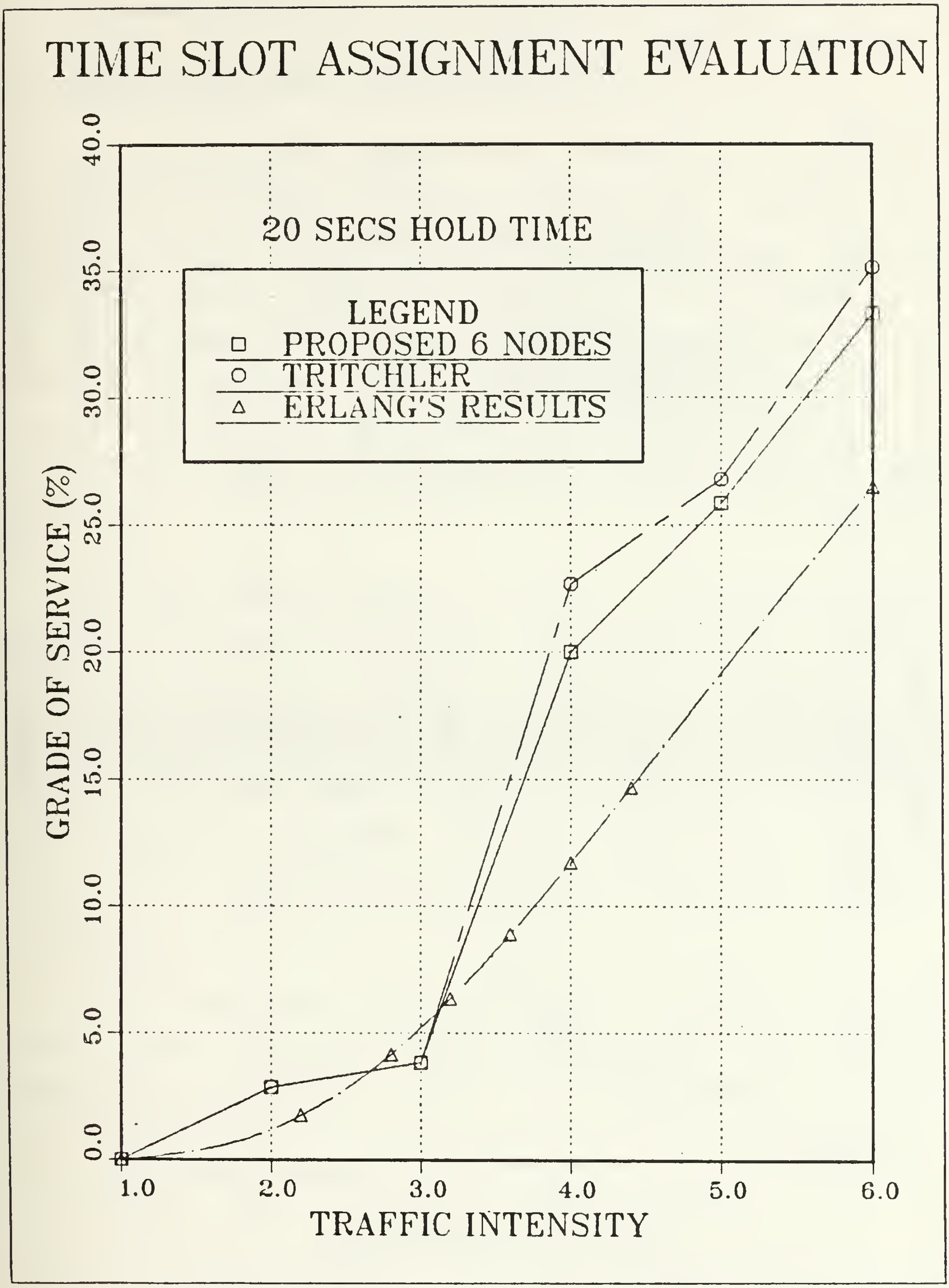


Figure 6.3 Results for 6 Nodes Network

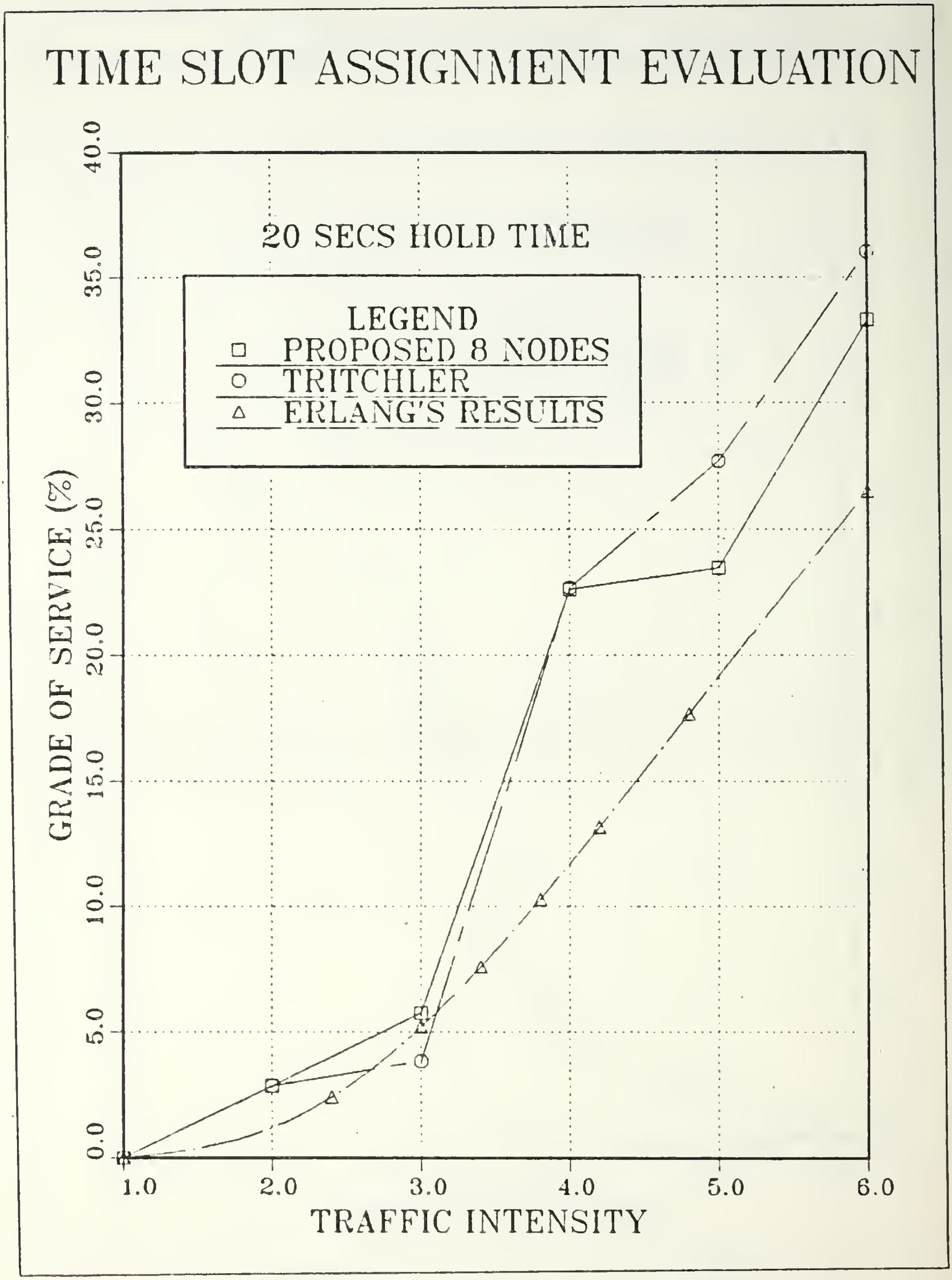


Figure 6.4 Results for 8 Nodes Network

B. RESULTS FOR STATIC AND DYNAMIC CONTROL

One of the major functions assigned to the routing function is to route traffic around congested links. For appropriate updating periods, this will prevent the congestion from becoming worse and maintain high network throughput. However, since the alternate path is longer than the static least hop path, packets will consume more link capacity. In some conditions, especially when routing is not updated frequently enough, this may cause the congestion to increase and spread. This phenomena can be observed from Figure 6.5 . The corresponding results are contained in Table III .

When the updating period is 15 secs, the network always performs worse than static least hop scheme does. When the updating period is 5 secs, the network at lower traffic intensities always performs better than the static least hop scheme. This is especially true for the 2 secs updating period. This shows that the proposed distance function is able to produce desirable path assignments for achieving high network throughput by using appropriate updating periods, so long as the network is not too close to saturation.

This simulation as a whole has demonstrated the effectiveness of the proposed link weight function in handling both the dynamic network conditions and the static network conditions.

TABLE III

Results for Static and Dynamic Network Control using the Dijkstra Algorithm

Traffic intensity	Percentage of lost call			
	Static least hop	2 secs update	5 secs update	15 secs update
1	0.00	0.00	0.00	0.00
2	1.40	0.00	0.00	2.10
3	2.31	0.00	0.46	3.24
4	4.55	3.15	3.50	6.99
5	6.41	4.46	4.74	10.58
6	11.63	9.40	12.08	16.33
7	15.11	13.77	17.97	21.22

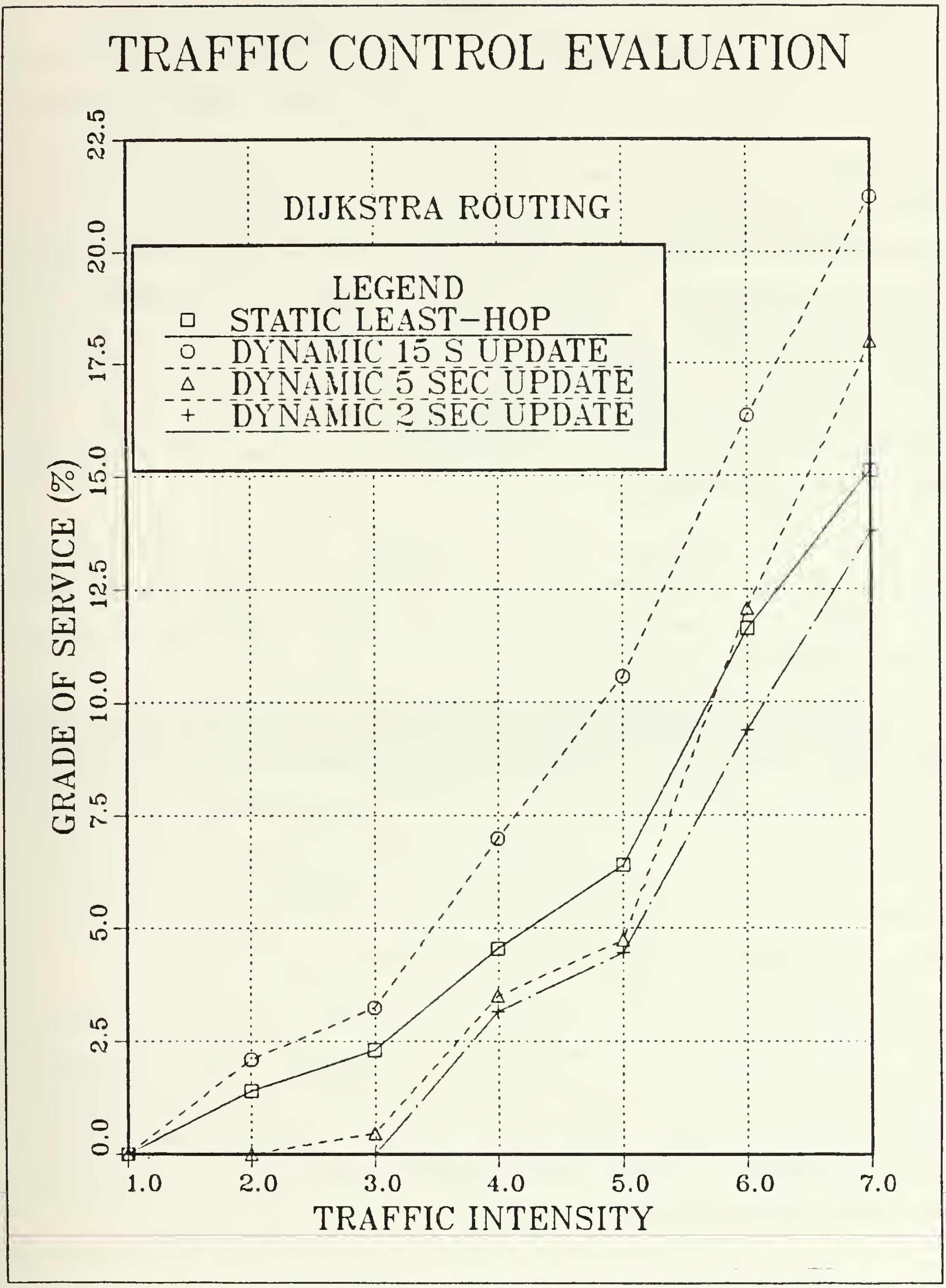


Figure 6.5 Results for Static and Dynamic Network Control

C. RESULTS FOR YEN ROUTING CONDITIONS

The Yen routing algorithm is employed with the proposed distance function to select paths that have minimum resistance to additional flows and achieve maximum network throughput. In the Yen algorithm, a node will receive routing information from all nodes in the network via its neighbors and immediately update its own routing table with the new information. In this manner, the routing table will contain the identity of the neighbor on the shortest path to any other nodes in the network. We note that the Yen routing algorithm is loop free. If the routing is not loop free, the network can become totally congested even when the entering traffic is not heavy.

It is worth noting that top priority for slot allocation is given to routing updates so that these messages will not get caught in network congestion and seriously affect network performance.

Simulation results given in Figure 6.6 and Table IV shows the network performance as functions of traffic intensity and update period. With an appropriate updating period, say 5 secs, the dynamic network control can balance the loading to prevent clustering of traffic. In this case, it shows a better performance than the static least-hop scheme for traffic intensities less than 4.3

We observe that the system acts like a least-hop scheme when the network is moderately loaded. For traffic intensities greater than 4.3, the static least-hop scheme performs better than the dynamic scheme. This indicates that routing messages under heavy traffic have great impact on the network performance.

TABLE IV

Results for Network Control using the Yen Routing

Traffic intensity	Grade of service			
	Static least hop	2 secs update	5 secs update	15 secs update
1	0.00	0.00	0.00	0.00
2	1.40	0.00	0.00	1.05
3	2.31	0.93	1.16	4.17
4	4.55	4.55	4.03	6.99
5	6.41	9.02	8.37	8.78
6	11.63	13.87	16.22	17.00
7	15.11	19.12	20.27	22.18

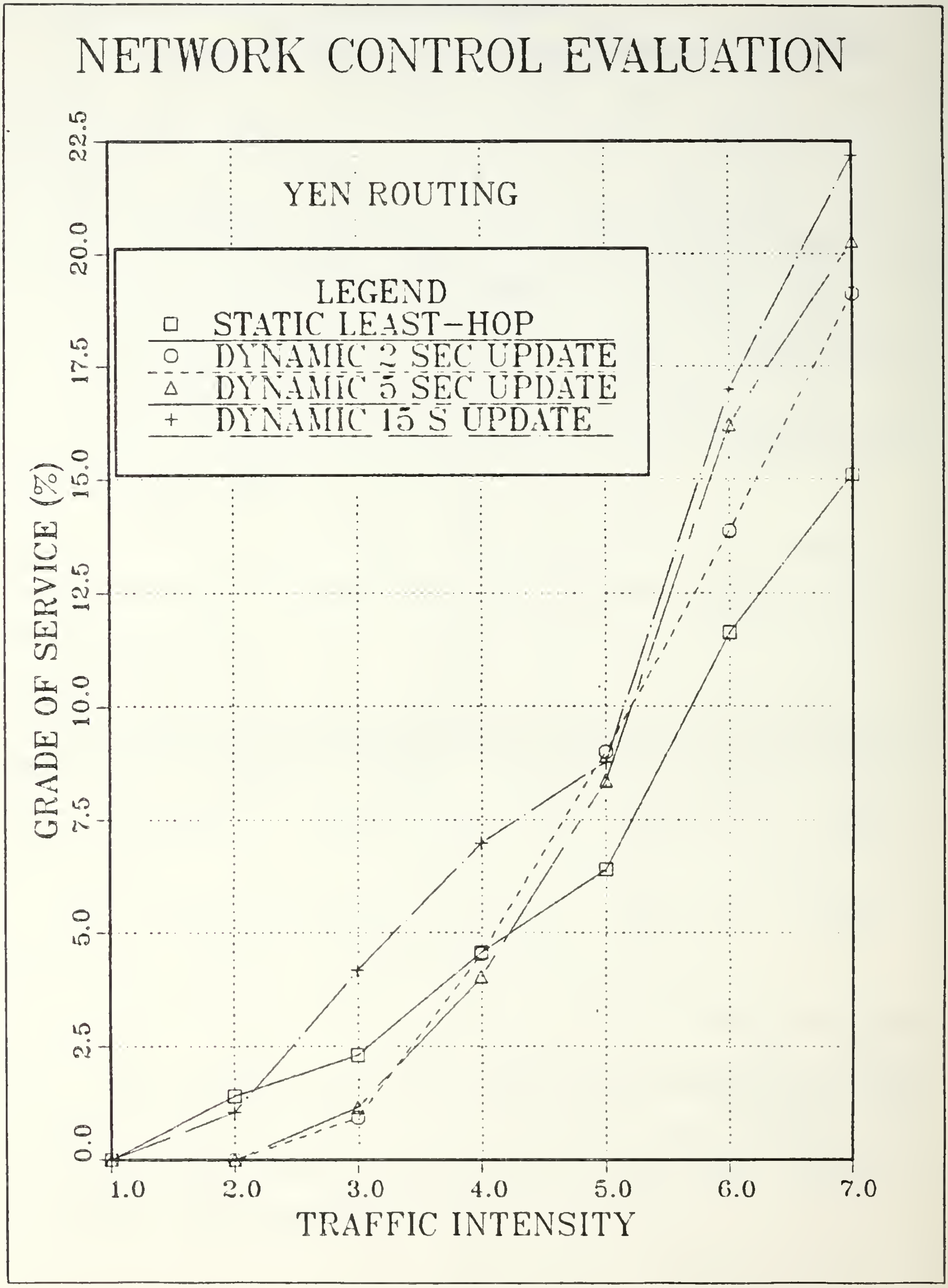


Figure 6.6 Results for Network Control using Yen Routing

VII. CONCLUSIONS

We have presented in this thesis an efficient and reliable packet radio network for military voice communications. A variety of network topologies for military applications was given. We noted that decentralized networks are commonly used in larger military radio networks. We also looked in detail at the network modeling and assumed that all the nodes are equally capable so that the packet radios are standardized and interoperable. This provides system flexibility, which is very important for military operations.

We have examined a packet virtual circuit technique for establishing voice sessions between source nodes and destination nodes. The key characteristics of this switched connection is that it allows voice packets to follow a fixed path with constant end-to-end delay through the network. It is worth mentioning that with voice packets we do not require retransmission of packets with errors; there will usually be less effect on voice intelligibility due to dropping bad packets, rather than incurring delay by retransmitting them.

To provide high channel quality and to satisfy traffic needs, a TDMA/CDMA hybrid scheme was proposed and evaluated. The unique features of this scheme are better utilization of channel, selective addressing capability for multiple users, simple processing in relaying messages, low probability of interception, anti-interference and antijam.

A simple time slot assignment algorithm which permits expeditious and efficient handling of call connections was proposed. We have demonstrated its performance by computer simulation. The results show that it is robust and stable with respect to the changes in network topology. A desirable slot depth (weighting complexity versus performance) was shown to be two.

We used Erlang's results to verify the simulation results and observed the major difference between wire line network and packet radio network. The results show that each node having eight possible channels could on the average provide service for six channels. This indicates that packet radio network performs somewhat worse than wire line network given same amount of channels under same traffic loads. One should realize at this point that a packet radio cannot receive and transmit at the same time but wire lines do.

Since the user demands are random in nature, a method for measuring the current state of the channel at each node is necessary. We found a method to estimate the unused transmission capacity of the inter-node links. An appropriate link weight function associated with this estimation was proposed. We conducted a series of simulation runs to investigate the behavior and performance of this function. The simulation results show that when a proper updating period is employed, the proposed function allows dynamic routing to produce desired path assignments for achieveing maximum network throughput. The results also show that when routing is not performed frequently enough, the network becomes unnecessary congested and affects traffic throughput because the old routing is used. This is especially true in the cases of heavy traffic when the traffic patterns change quickly.

We conclude that the proposed link weight function with an efficient dynamic routing algorithm using a proper updating period will maintain maximum network throughput at various traffic intensities before the network is heavily loaded.

APPENDIX A
THE DIJKSTRA SHORTEST PATH ALGORITHM

DIJKSTRA proposed a labeling algorithm to find the shortest path between two specific nodes in a graph in 1959. His algorithm requires one to know the graph topology and all distances between each pair of nodes.

Initially, no paths are known, all labels are tentative, so all nodes are labeled with infinity. As the algorithm proceeds, each node is labelled with its distance from the source node along the shortest path. When it is discovered that a label represents the shortest possible path from the source to that node, it is made permanent. This iterates until all of the mininum distance paths have been identified.

In this thesis, the DIJKSTRA algorithm is implemented as the successive calculation of

$$Dij = \text{Min} \{Dij \ , \ \underset{k}{\text{Min}} \ (Dik + Dkj)\}$$

for each pair of nodes in the network, where Dij represents the distance from node i to node j.

APPENDIX B
THE YEN SHORTEST PATH ALGORITHM

YEN J.Y. proposed a decentralized alogorithm for finding the shortest paths in communications networks in 1979. His algorithm requires little information exchanged between nodes and no global knowledge of the topology of the routes. The necessary assumptions for using the algorithm are as follows:

1. Each node in the network has processing capability and a timing device called clock, and knows correct time.
2. Each node sends its updated message independently.
3. Transit delay is neglected.

In an N-node directed network (see Figure B.1), we define (J,K) to be the link connecting node J to node K, YEN(J,K) to be the distance of the tentative shortest path from node J to node K, CLOCK(J,K) to be the time represents the corresponding value of YEN(J,K), and DISTANCE(J,I) to be the distance from node J to node I.

The following steps briefly describe his algorithm:

step 1 : Initially, all YEN(J,K)'s and CLOCK(J,K)'s are set to a large positive number (say 999999).

step 2 : At time 0, the destination node K sends each of its neighbor node a message "K".

step 3 : On receiving "K" from node I, each node J will

1. label the node (node I) that has just sent the message and delete it from its update transmission list.

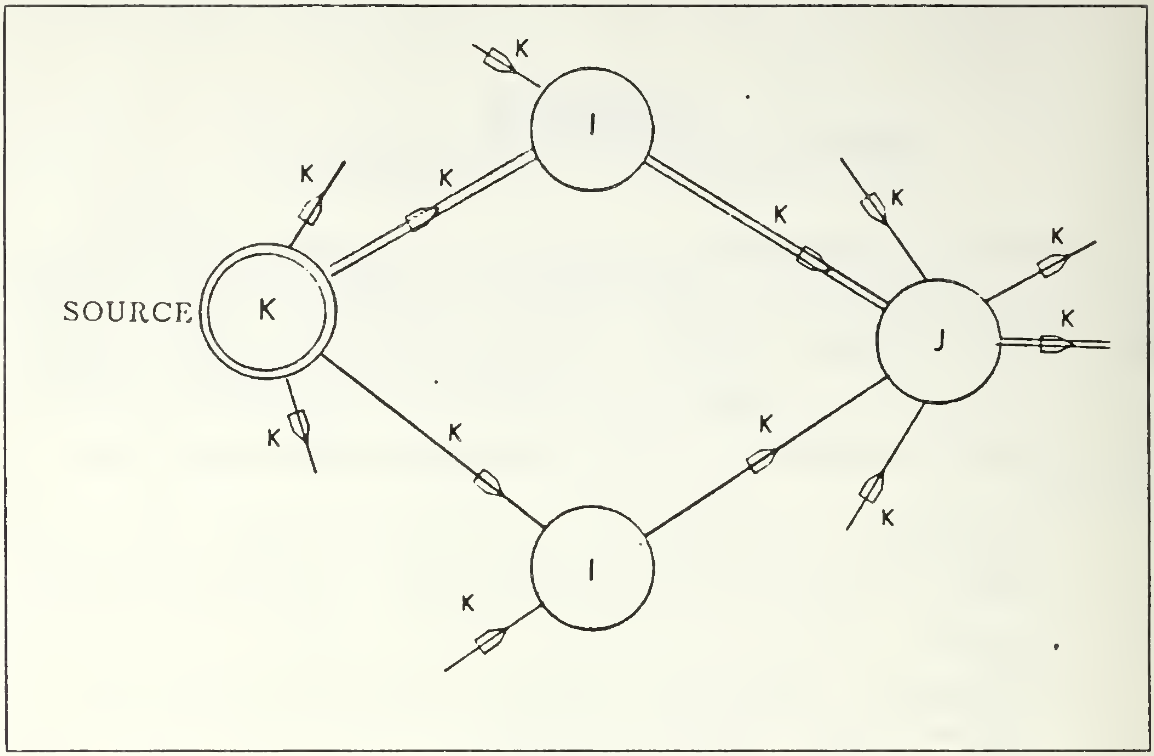


Figure B.1 Yen Routing Paths

2. update YEN(J,K) by
   YEN(J,K) = Min(YEN(I,K)+DISTANCE(J,I), YEN(J,K)) and set CLOCK(J,K) to the value of the updated YEN(J,K).
3. At time CLOCK(J,K), node J sends the message "K" to neighbors on its update transmission list.

step 4 : Repeat step 3 until message "K" has reached all the nodes in the network. This can be done by specifying a maximum possible time for message "K" to travel.

At the conclusion of the algorithm, each node J has the identity of the next node (node I) on the shortest path from node J to node K with the optimal shortest distance YEN(J,K).

APPENDIX C

ESTIMATION OF TRANSMISSION CAPACITY

The objective of estimating the available number of channels between two nodes in a network is to allow the distributed routing algorithm to produce desirable path assignments and achieve high throughput efficiency. In other words, when the overall traffic is not heavy, the algorithm reduces unnecessary local congestion due to statistical peaking of traffic intensities at some nodes and not the others. When more traffic enters the network, the dynamic routing uses the estimation result to distribute traffic evenly to maintain high throughput in the network. When excessive traffic enters the network, it spreads the congestion throughout the entire network.

As discussed in the thesis, a TDMA/CDMA scheme is used for each node to share a common broadcast RF frequency in packet radio networks. A virtual circuit is established for each voice conversation. Each virtual circuit needs a pair of time slots on each link and requires one link or a series of links from a calling node to a destination node. The slots associated with each virtual circuit will be reserved for the duration of the conversation.

Since every node is listening to its neighbors in any slot in which it is not transmitting, a node can infer a great deal about its neighbor's transmit slot assignment. This is the crucial point in the following derivation.

Consider a case where two received signals could be stacked in a slot and consider the following analysis. First of all, we define three terms: N_A, X_B and R_B. These quantities are measured at node A.

1. N_A = Number of empty slots at node A in which node B is not transmitting
2. X_B = Actual number of slots in which node B is transmitting
3. R_B = Number of slots in which node B is receiving

Without exchanging information with node B, node A does not know exactly about R_β value; it therefore estimates possible R_β values based on the X_β value. Assuming 12 slots per frame and 2 received signals could be stacked in a slot, then

1. $\left[\frac{X_B + 1}{2} \right] \le R_B \le X_B$ for $1 \le X_B \le 6$

2. $\left[\frac{X_B + 1}{2} \right] \le R_B \le 12 - X_B$ for $6 < X_B \le 8$

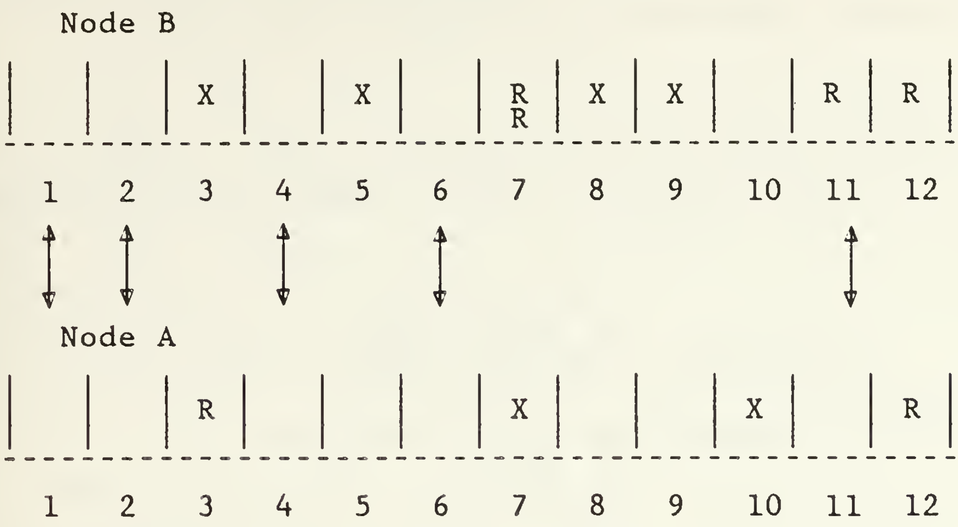
$[\quad] \triangleq$ Truncation to integer value

An example is given in Figure C.1 to clarify these definitions.

Have R_β slots uniformly distributed over $N_\beta = (12 - X_\beta)$ possible slots at node B, then

1. there are $\binom{N_B}{R_B}$ different patterns,

2. each pattern is equally likely.



$R \triangleq$ Receive $\quad X \triangleq$ Transmit

$X_B = 4$

$N_A = 5$

$R_B = 2, 3, 4$

In this example, $R_B = 3$

Figure C.1 Slot Distributions of Node B and Node A

Let Y be number of slots of set $\{R_\beta\}$ which overlap slots in set $\{N_A\}$,

if Y = 1, there are $\binom{N_A}{1}\binom{N_B - N_A}{R_B - 1}$ patterns

if Y = 2, there are $\binom{N_A}{2}\binom{N_B - N_A}{R_B - 2}$ patterns

In general, there exists $\binom{N_A}{Y}\binom{N_B - N_A}{R_B - Y}$ patterns.

The probability of Y slots in N_A (i.e. that Y receive slots at node B overlap the empty slots of node A) is hence given as

$$\text{Prob}\{Y / R_B, X_B, N_A\} = \begin{cases} \dfrac{\binom{N_A}{Y}\binom{N_B - N_A}{R_B - Y}}{\binom{N_B}{R_B}} , & 0 \le Y \le N_A \text{ and } \\ & N_A - (N_B - R_B) \le Y \le R_B \\ 0 , & \text{otherwise} \end{cases}$$

Since each pattern is equally probable, the average number of receive slots of node B which overlap slots at node A in the set $\{ N_A \}$ is given as

$$E\{Y/R_B, X_B, N_A\} = \sum_{\text{all } Y} Y \, \text{Prob}\{Y/R_B, X_B, N_A\}$$

If we average also over R_B (assuming each of its allowable values is equally likely), we have

$$\bar{Y} \triangleq E\{Y/X_B, N_A\} = \begin{cases} \dfrac{\Sigma \, E\{Y/R_B, X_B, N_A\}}{X_B - \left[\dfrac{X_B + 1}{2}\right]} & \text{for } 1 \le X_B \le 6 \\ \dfrac{\Sigma \, E\{Y/R_B, X_B, N_A\}}{X_B - \left[\dfrac{X_B + 1}{2}\right]} & \text{for } 6 \le X_B < 8 \end{cases}$$

Finally, the number of channels in which A can transmit to B, given X_β and N_A, is given by

$$C_{B/A} = N_A - \bar{Y}$$

It is worth noting that

$$C_{B/A} = C_{A/B} = K$$

does not imply that K circuits can be established.

This can be best explained by example (Figure C.2)

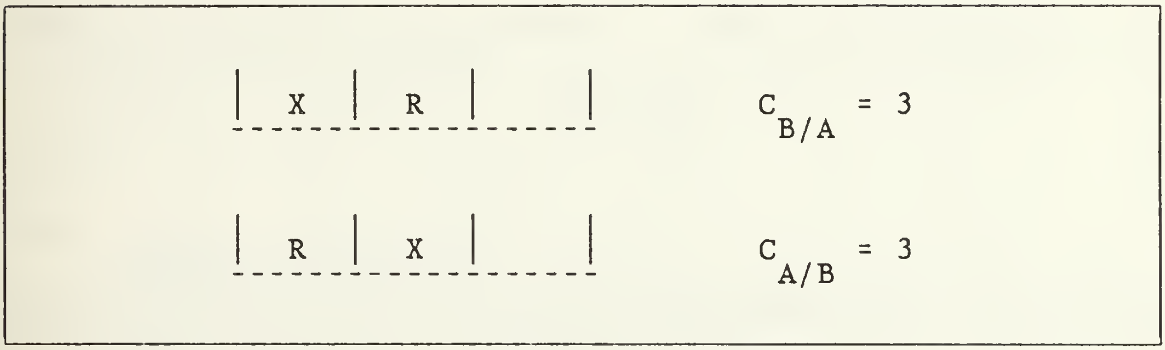


Figure C.2 Possible Arrangment for Circuits

Given $C_{B/A} = C_{A/B} = 3$, only one circuit could be established.

The link congestion level measured at node A from node A to node B can then be estimated by the following relationship

$$C.LEVEL(B,A) = Const / C_{B/A}$$

Where Const is an arbitrary constant.

This inverse relationship has a highly varying characteristic. When it is incorporated in the link weight, it imposes a high penalty to the congested link so that new path would be routed to bypass the congested nodes. In this manner, one can hope to maximize the system throughput and provide the desired grade of service.

C.LEVEL(B,A) indirectly measures the availablity or the transmission capacity of the link connecting node A to node B. Given any slot depth, it is always possible to perform the similiar computation for the C.LEVEL between a pair of nodes with any X_B and N_A by applying the same procedure.

APPENDIX D

SIMULATION PROGRAM FOR EVALUATING TIME SLOT ASSIGNMENT ALGORITHM

```
//SLOT   JOB (3060,0203),'SIMPLE',CLASS=C
//*MAIN ORG=NPGVM1.3060P
//  EXEC SIM25CLG,REGION.GO=4096K,PARM.GO='MAP,SIZE=760K'
//SIM.SYSIN DD *
''
PREAMBLE
''
NORMALLY MODE IS INTEGER
''
GENERATE LIST ROUTINES
''
TEMPORARY ENTITIES.......
 EVERY MESSAGE HAS A CKT.NUMBER, A TYPE, AN ORIGINATOR,
  A DESTINATION, A FM.NODE,
  A TO.NODE, A START.TIME, A SLOT.ARRIVAL, A SLOT.ASSIGN,
  A RECSLOT, A DIRECTION, A REATTEMPT
''
 DEFINE START.TIME AS A REAL VARIABLE
''
''
EVENT NOTICES INCLUDE REQUEST.FOR.SVC,
      RESPONSE.TO.REQUEST,
      UPSTREAM.BREAK.DOWN, NEW.CALL,
      DOWNSTREAM.BREAK.DOWN AND HALT.SIMULATION
''
      EVERY REQUEST.FOR.SVC HAS A MSG1
      EVERY RESPONSE.TO.REQUEST HAS A MSG2
      EVERY UPSTREAM.BREAK.DOWN HAS A U.B.D.MSG
      EVERY DOWNSTREAM.BREAK.DOWN HAS A D.B.D.MSG
''
''
PRIORITY ORDER IS UPSTREAM.BREAK.DOWN,
         DOWNSTREAM.BREAK.DOWN,AND HALT.SIMULATION
''
''
DEFINE INFO AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE SPECINFO AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE SLOTS.PER.FRAME AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE SLOT.DEPTH AND N AS INTEGER VARIABLES
DEFINE CALLED.NODE AND CALLING.NODE AS INTEGER VARIABLES
DEFINE PRNT.COUNTER AS AN INTEGER VARIABLE
DEFINE CKT.GENERATED, CKT.ESTAB, CKT.FAILED, CKT.SUM
       AND CKT.DISESTAB AS INTEGER VARIABLES
DEFINE UP.ROUTE AND DOWN.ROUTE AS INTEGER VARIABLES
DEFINE PRNT AS AN INTEGER VARIABLE
DEFINE TEST.DURATION, SLOT.DURATION,
       MEAN.SYS.CALL.ARRIV,
       AND MEAN.CALL.DURATION AS REAL VARIABLES
DEFINE NODAL.MEAN.CKT.ESTAB AS A REAL VARIABLE
DEFINE FAIR.POINTER AS AN INTEGER VARIABLE
DEFINE LONG.TIME.EST, AVG.P.BD, LONG.P.BD, AVG.C.BD,
       LONG.C.BD AND
       AVG.TIME.EST AS REAL VARIABLES
DEFINE CKT.LONG.TIME.EST AS AN INTEGER VARIABLE
DEFINE MAX.CKT AS AN INTEGER VARIABLE
DEFINE SUM.BD.TIME, AVG.BD.TIME, TOT.P.BD AND TOT.C.BD
```

```
      AS REAL VARIABLES
DEFINE CKTS.BD AS AN INTEGER VARIABLE
DEFINE FRACT.LOST.CALL AND FRACT.SUCCESSFUL.CALL
       AS  REAL VARIABLES
DEFINE C.BD.COUNTER AND P.BD.COUNTER
       AS INTEGER VARIABLES
DEFINE SUM.DURATION AND CALL.DURATION AS REAL VARIABLES
DEFINE DELAY.SUM AND AVG.DURATION AS REAL VARIABLES
DEFINE PRNT.INTERVAL AS A REAL VARIABLE
DEFINE OFFERED.TRAFFIC AS AN INTEGER VARIABLE
DEFINE BREAKTIME AS A REAL VARIABLE
DEFINE MAX.ATTEMPT AS AN INTEGER VARIABLE
DEFINE PARTIAL.BREAKDOWN TO MEAN 3
DEFINE FULL.BREAKDOWN TO MEAN 4
''
END ''PREAMBLE
''
''
''
''   &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
MAIN
''
LET LINE.V = 80
START NEW PAGE
''
PRINT 5 LINES AS FOLLOWS
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$                                                            $
$          RESULTS OF SIMULATION                             $
$                                                            $
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
SKIP 4 OUTPUT LINES
''
''
''   THE MAIN PROGRAM CALLS INPUT.DATA ROUTINE THAT SETS
''   INPUT PARAMETERS AND INTIALIZATION VARIABLES
''   FOR ALL RUNS OF SIMULATIONS.
''
''
LET PRNT.COUNTER = 0
LET FAIR.POINTER = 1
''
PERFORM FRESH.INPUT
RESERVE INFO(*,*,*) AS (20 + N ) BY 12 BY 4
RESERVE SPECINFO(*,*) AS (20 + N) BY 12
RELEASE SEED.V(*)
RESERVE SEED.V(*) AS 10
''
LET SEED.V(1) = 2116429302
LET SEED.V(2) =  683743814
LET SEED.V(3) =  964393174
LET SEED.V(4) = 1217426631
LET SEED.V(5) =  618433579
LET SEED.V(6) = 1157240309
LET SEED.V(7) =   15726055
LET SEED.V(8) =   48108509
LET SEED.V(9) = 1797920909
LET SEED.V(10) = 477424540
''
''
''
PRINT 2 LINES WITH ((N + 1) * 2)
      AND SLOT.DEPTH AS FOLLOWS
THE NUMBER OF NODES IN THE SYSTEM ARE **
THE SLOT DEPTH NOW IS **
SKIP 2 OUTPUT LINES
''
''
```

```
'' INFO(NODE,SLOT,INDEX) = INTEGER VALUE
''
''    NODE DENOTES NODE NUMBER
''    SLOT DENOTES SLOT NUMBER
''    INDEX = 0  : EMPTY SLOT
''    INDEX = 1  : TRANSMIT'S SLOT WITH CIRCUIT NUMBER
''    INDEX = 2  : RECEIVE'S SLOT
''    INDEX = 3  : NODE NUMBER
''    INDEX = 4  : NUMBER OF RECEIVE SIGNALS
''
''
SCHEDULE A NEW.CALL NOW
SCHEDULE A HALT.SIMULATION AT TEST.DURATION
''
START SIMULATION
''
SKIP 2 OUTPUT LINES
PRINT 1 LINE AS FOLLOWS
THE PROGRAM HAS COME TO THE END OF THE SIMULATION
''
STOP
END
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR FRESH.INPUT
''
'' THIS ROUTINE SETS ALL INPUT VARIABLES IN ORDER TO
'' BEGIN THE SIMULATION
''
''
LET PRNT = 5
''
'' PRNT HELPS IN DEBUGGING THE SOFTWARE AND PROGRAM LOGIC
''
''  0  == PRINT  EACH PROCESS
''  1  == PRINT SLOT ASSIGNMENT AT EACH NODE
''  2  == SELECTIVE PRINTING
''
'' INPUT DATA
''
LET OFFERED.TRAFFIC = 6
LET N = 3
LET SLOT.DEPTH = 2
LET MAX.ATTEMPT = 4
LET TEST.DURATION= 400.00
LET SLOT.DURATION= 0.000010417
LET MEAN.CALL.DURATION = 20.0000
LET NODAL.MEAN.CKT.ESTAB = (MEAN.CALL.DURATION * 2.0) /
                           REAL.F(OFFERED.TRAFFIC)
LET MAX.CKT = 300
LET PRNT.INTERVAL = 30.00000
''
''
'' INITIALIZATION
''
LET TIME.V = 0.000000000
LET CKT.GENERATED = 0
LET CKT.DISESTAB = 0
LET CKT.SUM = 0
LET CKT.ESTAB = 0
LET MEAN.SYS.CALL.ARRIV = NODAL.MEAN.CKT.ESTAB / 2.0
LET FRACT.SUCCESSFUL.CALL = 0.0
LET FRACT.LOST.CALL = 0.0
LET UP.ROUTE = 0
LET DOWN.ROUTE = 0
LET BREAKTIME = 13.0 * SLOT.DURATION
```

```
LET SUM.DURATION = 0.0
LET AVG.DURATION = 0.0
LET LONG.TIME.EST = 0.0
LET AVG.TIME.EST = 0.0
LET AVG.P.BD = 0.0
LET AVG.C.BD = 0.0
LET LONG.C.BD = 0.0
LET LONG.P.BD = 0.0
LET CKT.LONG.TIME.EST = 0.0
LET AVG.BD.TIME = 0.0
LET SUM.BD.TIME = 0.0
LET CKTS.BD = 0
LET P.BD.COUNTER = 0
LET C.BD.COUNTER = 0
LET TOT.P.BD = 0.0
LET TOT.C.BD = 0.0
LET CKT.FAILED = 0
LET AVG.DURATION = 0.0
LET DELAY.SUM = 0.0
''
PRINT 5 LINES WITH TEST.DURATION, SLOT.DURATION,
                   NODAL.MEAN.CKT.ESTAB,
                   MEAN.CALL.DURATION, AND PRNT.INTERVAL
                   AS FOLLOWS
THE SIMULATION WILL RUN FOR  ***.** SECS
THE DURATION OF A TIME SLOT IS *.******** SECS
THE MEAN GENERATION TIME OF A NEW CALL ***.** SECS
THE MEAN DURATION TIME OF A CIRCUIT IS ****.** SECS
THE PROGRAM WILL PRINT RESULTS EVERY ****.*** SECS
SKIP 2 OUTPUT LINES
''
''
RETURN
END '' FRESH.INPUT
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
''
EVENT NEW.CALL SAVING THE EVENT NOTICE
''
''
'' THIS EVENT GENERATES CALL AND SENDS REQUEST FOR
'' service FROM A CALLING NODE TO A CALLED NODE
''
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
NEW CALL GENERATED AT TIME ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
DEFINE DELAY1.PROB,AGGREGATE.PROB AS REAL VARIABLES
DEFINE ORG.NODE, AGGREGATE.NODE AND DEST.NODE
       AS INTEGER VARIABLES
''
''
'' N IS THE NUMBER OF AGGREGATE NODES CONNECTED TO
'' a node
''
LET CKT.GENERATED = CKT.GENERATED + 1
LET CKT.SUM = CKT.SUM + 1
''
IF CKT.SUM GE MAX.CKT
 PRINT 2 LINES WITH TIME.V AND MAX.CKT AS FOLLOWS
NUMBER OF CKTS ATTEMPTED EXCEEDS THE TOTAL NO OF CKTS
```

```
**** PERMITTED .  SIMULATION HALTED AT ****.*** SEC
''SKIP 1 OUTPUT LINE

 PERFORM TERMINATION
 RETURN
ALWAYS
''
SCHEDULE A NEW.CALL AT
time.v + EXPONENTIAL.F(MEAN.SYS.CALL.ARRIV,5)
''
'' SELECT A TRANSMIT NODE

LET ORG.NODE = RANDI.F(1,2,1)
''
''
LET PROB = UNIFORM.F(0.0,100.0,6)
LET AGGREGATE.PROB = REAL.F(N) * 100.00 / REAL.F(N + 1)
''
'' SELECT A RECEIVE NODE

IF PROB GE AGGREGATE.PROB
  IF ORG.NODE EQ 1
    LET DEST.NODE = 2
    LET CALLED.NODE = 2
  ALWAYS
  IF ORG.NODE EQ 2
    LET DEST.NODE = 1
    LET CALLED.NODE = 1
  ALWAYS
  GO TO OUT.NODE
ALWAYS
''
''LET AGGREGATE.NODE = RANDI.F(1,N,2)

 IF ORG.NODE EQ 1
   LET DEST.NODE = 20 + AGGREGATE.NODE
   LET CALLED.NODE = 2
 ALWAYS
 IF ORG.NODE EQ 2
   LET DEST.NODE = 10 + AGGREGATE.NODE
   LET CALLED.NODE = 1
''ALWAYS

'OUT.NODE'
''
 PRINT 1 LINE WITH CKT.SUM,ORG.NODE,DEST.NODE
                    AND TIME.V AS FOLLOWS
CIRCUIT *** FROM NODE ** TO NODE ** BEGUN AT ***.****
''SKIP 2 OUTPUT LINE

LET UP.ROUTE = UP.ROUTE + 1
''
FOR J = 1 TO 12 , DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 GO TO 'ON1'
ALWAYS
''
LOOP
''
IF PRNT EQ 0
 PRINT 3 LINES WITH ORG.NODE,CALLED.NODE AND CKT.SUM
                     AS FOLLOWS
NO MUTUALLY AVAILABLE SLOTS BETWEEN ORIG.NODE **
AND CALLED NODE ** TO CARRY THE REQUEST SERVICE MESSAGE
FOR CIRCUIT NUMBER *****
 SKIP 1 OUTPUT LINE
ALWAYS
''
```

```
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
GO TO LAST.NEW.CALL
''
'' SELECT A CURRENT SLOT RANDOMLY
'' AND CONTINUE PROCESSING
''
'ON1'
''
LET CURRENT.SLOT = RANDI.F(1,12,3)
''
''
'' FINDS THE NEXT MUTUALLY AVAILABLE SLOT
''
''
LET SLOT1 = 0
LET FRAME1 = 0
''
IF CURRENT.SLOT EQ 12
 GO TO NEXT.FRAME1
ALWAYS
''
LET K = CURRENT.SLOT + 1
''
FOR J = K TO 12 , DO
IF INFO(ORG.NODE,J,1) NE 0 OR INFO(ORG.NODE,J,4) NE 0
 LET SPECINFO(ORG.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,J) EQ 6
    AND INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
LET FRAME1 = 1
FOR J = 1 TO CURRENT.SLOT , DO
IF INFO(ORG.NODE,J,1) NE 0 OR INFO(ORG.NODE,J,4) NE 0
 LET SPECINFO(ORG.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,J) EQ 6 AND
   INFO(CALLED.NODE,J,1) EQ 0 AND
   INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
LET FRAME1 = 0
FOR J = K TO 12 , DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
''
LET FRAME1 = 1
FOR J = 1 TO CURRENT.SLOT, DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
```

```
GO TO YY
''
'NEXT.FRAME1'
''
LET FRAME1 = 1
FOR J = 1 TO 12, DO
IF INFO(ORG.NODE,J,1) NE 0 OR INFO(ORG.NODE,J,4) NE 0
 LET SPECINFO(ORG.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,J) EQ 6
    AND INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
''
'YY'
''
PRINT 1 LINE WITH CKT.SUM AS FOLLOWS
CIRCUIT NO.**** FAILED IN EVENT NEW CALL
SKIP 2 OUTPUT LINES
''
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
GO TO LAST.NEW.CALL
''
'' ON2 IDENTIFIES A SLOT TO CARRY THE SERVICE MESSAGE
'' TO THE CALLED NODE AND CREATES THE SERVICE MESSAGE
''
'ON2'
''
CREATE A MESSAGE
''
LET CKT.NUMBER(MESSAGE) = CKT.SUM
LET TYPE(MESSAGE) = 1
LET ORIGINATOR(MESSAGE) = ORG.NODE
LET DESTINATION(MESSAGE) = DEST.NODE
LET FM.NODE(MESSAGE) = ORG.NODE
LET TO.NODE(MESSAGE) = CALLED.NODE
LET START.TIME(MESSAGE) = TIME.V
LET SLOT.ARRIVAL(MESSAGE) = SLOT1
LET SLOT.ASSIGN(MESSAGE) = SLOT1
LET RECSLOT(MESSAGE) = SLOT1
LET DIRECTION(MESSAGE) = 0
LET REATTEMPT(MESSAGE) = 1
''
IF PRNT EQ 0
 PRINT 2 LINES WITH SLOT1 AND FRAME1 AS FOLLOWS
SLOT ** OF FRAME ** WAS USED TO CARRY REQUEST FOR SERVICE
FROM CALLING NODE TO THE CALLED NODE
 SKIP 1 OUTPUT LINE
ALWAYS
''
'' CALCULATES WHEN THE SERVICE MESSAGE WILL ARRIVE AT
'' THE CALLED NODE AND SCHEDULES ITS ARRIVAL
''
LET DELAY1 = (REAL.F(12 * FRAME1 + SLOT1 - CURRENT.SLOT))
             * SLOT.DURATION
```

```
''
IF PRNT EQ 0
 PRINT 2 LINES WITH CKT.SUM,CALLED.NODE AND
                (TIME.V + DELAY1) AS FOLLOWS
CIRCUIT NO. ***** HAS SCHEDULED AN REQUEST FOR SVC
AT NODE ** AT TIME ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
         AT TIME.V + DELAY1
''
'LAST.NEW.CALL'
''
RETURN
END '' NEW.CALL
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT REQUEST.FOR.SVC GIVEN MSG1
''
'' THIS EVENT SIMULATES ACTIONS PERFORMED AT A CALLED
'' node AFTER RECEIVING AN REQUEST FOR SERVICE FROM A
'' calling NODE
''
LET MESSAGE = MSG1
''
IF  PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
  REQUEST.FOR.SVC PERFORMED AT TIME  ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF PRNT EQ 0
 PRINT 2 LINE AS FOLLOWS
ATTRIBUTES OF MESSAGE ENTITY AT
THE START OF REQUEST.FOR.SVC ARE:
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
''
ALWAYS
''
DEFINE DELAY2 AND DELAYR AS  REAL VARIABLES
''
LET FRAME.REC = 0
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
LET SLOT.REC = CURRENT.SLOT
''
IF INFO(CALLED.NODE,SLOT.REC,4) LT SLOT.DEPTH AND
   INFO(CALLED.NODE,SLOT.REC,1) EQ 0
   GO TO OK1
ALWAYS
''
IF REATTEMPT(MESSAGE) LT MAX.ATTEMPT
 LET REATTEMPT(MESSAGE) = REATTEMPT(MESSAGE) + 1
 LET SLOT.USED = SLOT.REC
''
 LET FRAMER = 1
 FOR IR = (SLOT.USED + 1) TO 12, DO
 IF INFO(CALLING.NODE,IR,1) NE 0 OR
    INFO(CALLING.NODE,IR,4) NE 0
  LET SPECINFO(CALLING.NODE,IR) = 0
 ALWAYS
  IF SPECINFO(CALLING.NODE,IR) EQ 6 AND
     INFO(CALLED.NODE,IR,1) EQ 0 AND
     INFO(CALLING.NODE,IR,1) EQ 0 AND
```

```
      INFO(CALLING.NODE,IR,4) EQ 0
    LET SLOTR = IR
    LET SPECINFO(CALLING.NODE,IR) = 0
    GO TO MORE.ATTEMPT
   ALWAYS
  LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR JR = 1 TO (SLOT.USED - 1), DO
 IF INFO(CALLING.NODE,JR,1) NE 0 OR
    INFO(CALLING.NODE,JR,4) NE 0
  LET SPECINFO(CALLING.NODE,JR) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,JR) EQ 6 AND
    INFO(CALLED.NODE,JR,1) EQ 0 AND
    INFO(CALLING.NODE,JR,1) EQ 0 AND
    INFO(CALLING.NODE,JR,4) EQ 0
  LET SLOTR = JR
  LET SPECINFO(CALLING.NODE,JR) = 0
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
 LET FRAMER = 1
 FOR IR = (SLOT.USED + 1) TO 12, DO
 IF INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4)
    EQ 0 AND INFO(CALLED.NODE,IR,1) EQ 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR JR = 1 TO (SLOT.USED - 1), DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
     INFO(CALLING.NODE,JR,4)
     EQ 0 AND INFO(CALLED.NODE,JR,1) EQ 0
   LET SLOTR = JR
   GO TO MORE.ATTEMPT
  ALWAYS
 LOOP
 GO TO XX
''
'MORE.ATTEMPT'
''
 LET DELAYR = (REAL.F(SLOTR - SLOT.USED) + FRAMER * 12.0)
                * SLOT.DURATION
 LET RECSLOT(MESSAGE) = 0
 LET DIRECTION(MESSAGE) = 0
 LET SLOT.ASSIGN(MESSAGE) = SLOTR
 LET SLOT.ARRIVAL(MESSAGE) = SLOTR
''
 SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
          AT TIME.V + DELAYR
''
 RETURN
ALWAYS
''
'XX'
''
```

```
IF PRNT EQ 0
PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
TIME.V, TO.NODE(MESSAGE) AND FM.NODE(MESSAGE)
               AS FOLLOWS
CIRCUIT **** FR NODE ** TO ** BROKEDOWN AT ****.******
DUE TO NO MUTUALLY AVAILABLE SLOT BETWEEN THE CALLED
NODE ** AND THE CALLING NODE **
SKIP 2 OUTPUT LINE
ALWAYS
''
'EXIT'
''
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
''
IF FM.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
 LET P.BD.COUNTER = P.BD.COUNTER + 1
 DESTROY THE MESSAGE CALLED MESSAGE
 RETURN
ALWAYS
''
LET DOWN.ROUTE = DOWN.ROUTE + 1
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET DIRECTION(MESSAGE) = 3
LET START.TIME(MESSAGE) = TIME.V
''
 SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE
 AT TIME.V + BREAKTIME
 RETURN
''
'' FIND THE NEXT MUTUALLY AVAILABLE SLOT
''
'OK1'
''
LET SLOT2 = 0
LET FRAME2 = 0
''
IF CURRENT.SLOT EQ 12
 GO TO NEXT.FRAME2
ALWAYS
''
LET L = CURRENT.SLOT + 1
''
FOR J = L TO 12 , DO
IF INFO(CALLED.NODE,J,1) NE 0 OR
   INFO(CALLED.NODE,J,4) NE 0
 LET SPECINFO(CALLED.NODE,J) = 0
ALWAYS
IF SPECINFO(CALLED.NODE,J) EQ 6 AND
   INFO(CALLING.NODE,J,1) EQ 0 AND
   INFO(CALLED.NODE,J,1) EQ 0 AND
   INFO(CALLED.NODE,J,4) EQ 0
 LET SPECINFO(CALLED.NODE,J) = 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
''
LET FRAME2 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLED.NODE,J,1) NE 0 OR
    INFO(CALLED.NODE,J,4) NE 0
  LET SPECINFO(CALLED.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0
```

```
  LET SPECINFO(CALLED.NODE,J) = 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
''
LET FRAME2 = 0
FOR J = L TO 12, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
''
LET FRAME2 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
GO TO YYY
''
'NEXT.FRAME2'
''
LET FRAME2 = 1
FOR J = 1 TO 12, DO
 IF INFO(CALLED.NODE,J,1) NE 0 OR
    INFO(CALLED.NODE,J,4) NE 0
  LET SPECINFO(CALLED.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,J) = 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
''
'YYY'
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** FAILED IN EVENT REQUEST.FOR.SVC
SKIP 1 OUTPUT LINE
''
GO TO EXIT
''
'' OK2 IDENTIFIES A SLOT TO CARRY THE SLOT ASSIGNMENT
'' AND SENDS REQUEST BACK TO THE CALLING NODE AND
'' ALSO COMPUTES WHEN THE SERVICE MESSAGE WILL ARRIVE
'' AT THE CALLING NODE
''
'OK2'
''
```

```
LET DELAY2 = (REAL.F(12 * FRAME2 + SLOT2 - CURRENT.SLOT))
               * SLOT.DURATION
''
'' ASSIGNS SLOTS,UPDATES MESSAGE,AND
'' SCHEDULES RESPONSE.TO.REQUEST
'' AT THE CALLING NODE
''
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT2
LET SLOT.ASSIGN(MESSAGE) = SLOT.REC
LET RECSLOT(MESSAGE) = SLOT.REC
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE), FM.NODE(MESSAGE)
            AND (TIME.V + DELAY2) AS FOLLOWS
CIRCUIT **** HAS SCHEDULED A RESPONSE TO SVC AT NODE **
AT TIME ****.****
SKIP 2 OUTPUT LINES
''
PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT END OF REQUEST FOR SVC ARE:
LIST ATTRIBUTES OF MESSAGE
SKIP 1 OUTPUT LINE
ALWAYS
''
SCHEDULE A RESPONSE.TO.REQUEST GIVEN MESSAGE
AT TIME.V + DELAY2
''
RETURN
END '' REQUEST FOR SERVICE
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT RESPONSE.TO.REQUEST GIVEN MSG2
''
'' THIS EVENT SIMULATES ACTIONS PERFORMED AT A CALLING
'' NODE AFTER RECEIVING AN RESPONSE TO REQUEST
'' FROM A CALLED NODE
''
''
LET MESSAGE = MSG2
''
IF  PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 RESPONSE.TO.REQUEST PERFORMED AT TIME  ****.*****
''
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF PRNT EQ 0
 PRINT 2 LINE AS FOLLOWS
 ATTRIBUTES OF MESSAGE ENTITY
 AT THE START OF RESPONSE.TO.REQUEST ARE:
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
''
ALWAYS
''
DEFINE DELAY3 AND DELAYR AS REAL VARIABLES
''
LET FRAME.REC = 0
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
LET SLOT.REC = SLOT.ARRIVAL(MESSAGE)
''
IF INFO(CALLING.NODE,SLOT.REC,1) EQ 0
```

```
   AND INFO(CALLING.NODE,SLOT.REC,4) LT SLOT.DEPTH
 GO TO CORRECT
 ALWAYS
''
IF REATTEMPT(MESSAGE) LT MAX.ATTEMPT
 LET REATTEMPT(MESSAGE) = REATTEMPT(MESSAGE) + 1
 LET SLOT.USED = RECSLOT(MESSAGE)
''
 LET FRAMER = 1
 FOR IR = (SLOT.USED + 1) TO 12, DO
 IF INFO(CALLING.NODE,IR,1) NE 0 OR
    INFO(CALLING.NODE,IR,4) NE 0
  LET SPECINFO(CALLING.NODE,IR) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,IR) EQ 6 AND
    INFO(CALLED.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4) EQ 0
  LET SPECINFO(CALLING.NODE,IR) = 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR IR = 1 TO (SLOT.USED - 1), DO
 IF INFO(CALLING.NODE,IR,1) NE 0 OR
    INFO(CALLING.NODE,IR,4) NE 0
  LET SPECINFO(CALLING.NODE,IR) = 0
 ALWAYS
''
 IF SPECINFO(CALLING.NODE,IR) EQ 6 AND
    INFO(CALLED.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4) EQ 0
  LET SPECINFO(CALLING.NODE,IR) = 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
 LET FRAMER = 1
 FOR JR = (SLOT.USED + 1) TO 12, DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
     INFO(CALLING.NODE,JR,4) EQ 0 AND
     INFO(CALLED.NODE,JR,1) EQ 0
   LET SLOTR = JR
   GO TO MORE.ATTEMPT
  ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR JR = 1 TO (SLOT.USED - 1), DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
     INFO(CALLING.NODE,JR,4)
     EQ 0 AND INFO(CALLED.NODE,JR,1) EQ 0
   LET SLOTR = JR
   GO TO MORE.ATTEMPT
  ALWAYS
 LOOP
 GO TO XX
```

```
''
'MORE.ATTEMPT'
''
 LET DELAYR = (REAL.F(SLOTR - SLOT.REC)+(FRAMER * 12.0))
                 * SLOT.DURATION
 LET RECSLOT(MESSAGE) = 0
 LET SLOT.ASSIGN(MESSAGE) = 0
 LET DIRECTION(MESSAGE) = 0
 LET SLOT.ARRIVAL(MESSAGE) = SLOTR
''
 SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
          AT TIME.V + DELAYR
''
 RETURN
ALWAYS
''
'XX'
''
IF PRNT EQ 0
PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
TIME.V, FM.NODE(MESSAGE) AND
TO.NODE(MESSAGE) AS FOLLOWS
CIRCUIT **** FROM NODE ** TO ** BROKEDOWN AT ***.***** DUE
TO NO MUTUALLY AVAILABLE SLOT BETWEEN THE CALLED NODE **
AND THE CALLING NODE **
SKIP 2 OUTPUT LINES
ALWAYS
''
''  SCHEDULES BREAK-DOWN TO BEGIN  AT THE CALLED NODE
''  AFTER A DELAY OF 13 * SLOT DURATION UNITS TO SIMULATE
''  "TIME OUT" IF NO SLOT IS AVAILABLE TO
''  CARRY THE COORDINATION MESSAGE
''
''
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET DOWN.ROUTE = DOWN.ROUTE + 1
''
IF PRNT EQ 0
 PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AND TIME.V
                    AS FOLLOWS
CIRCUIT NO.***** FAILED TO CONNECT AT ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF FM.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
 LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
 LET START.TIME(MESSAGE) = TIME.V
 LET RECSLOT(MESSAGE) = 13
 LET DIRECTION(MESSAGE) = 0
 SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN MESSAGE AT TIME.V
                                                 + BREAKTIME
 RETURN
ALWAYS
''
 LET DIRECTION(MESSAGE) = 4
 SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE NOW
''
 RETURN
''
''
'CORRECT'
''
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),1) =
                           CKT.NUMBER(MESSAGE)
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),2) =
                                      SLOT.REC
```

```
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),3) =
                                    CALLED.NODE
LET INFO(CALLING.NODE,SLOT.REC,4) =
INFO(CALLING.NODE,SLOT.REC,4) + 1
LET XSLOT.CALLED = SLOT.REC
LET RSLOT.CALLED = SLOT.ASSIGN(MESSAGE)
LET SLOT.ARRIVAL(MESSAGE) = RSLOT.CALLED
LET SLOT.ASSIGN(MESSAGE) = XSLOT.CALLED
LET RECSLOT(MESSAGE) = XSLOT.CALLED
''
LET INFO(CALLED.NODE,XSLOT.CALLED,1) =
                                CKT.NUMBER(MESSAGE)
LET INFO(CALLED.NODE,XSLOT.CALLED,2) = RSLOT.CALLED
LET INFO(CALLED.NODE,XSLOT.CALLED,3) = CALLING.NODE
LET INFO(CALLED.NODE,RSLOT.CALLED,4) =
               INFO(CALLED.NODE,RSLOT.CALLED,4) + 1
''
''
'' CHECK WHETHER THE CIRCUIT IS COMPLETE
'' IF YES, CALL THE COMPLETE.CKT ROUTINE AND
'' COLLECT STATISTICAL DATA
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM VIRTUAL.CKT GIVEN MESSAGE
 RETURN
ALWAYS
''
'' IF THE CIRCUIT HAS NOT BEEN ESTABLISHED ALL THE WAY
'' TO THE DESTINATION ,THEN SPECIAL ACTION MUST BE TAKEN
'' TO ESTABLISH THE NEXT LINK TO THE DESTINATION
''
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
LET TO.NODE(MESSAGE) = DESTINATION(MESSAGE)
''
'' THE REST OF THIS EVENT SIMULATES ACTIONS PERFORMED AT
'' AN INTERMEDIATE NODE .
''
'' WE BEGIN TO CHECK WHETHER THERE IS A SLOT AVAILABLE AT
'' THIS NEWLY ASSIGNED CALLING NODE TO ACCOMODATE
'' THE TRANSMISSION TO THE NEWLY
'' ASSIGNED CALLED NODE.
''
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
''
'' THE CURRENT SLOT IS CONTAINED IN THE  MESSAGE ATTRIBUTE
'' SLOT ARRIVAL
''
'' FIND THE NEXT MUTUALLY AVAILABLE SLOT
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET SLOT3 = 0
LET FRAME3 = 0
''
IF CURRENT.SLOT EQ 12
 LET K = 1
 GO TO NEXT.FRAME3
ALWAYS
''
LET K = CURRENT.SLOT + 1
FOR J = K TO 12 , DO
 IF INFO(CALLED.NODE,J,1) NE 0 OR
    INFO(CALLING.NODE,J,4) NE 0
  LET SPECINFO(CALLING.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLED.NODE,J) EQ 6 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
```

```
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
  LET SPECINFO(CALLING.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
LET FRAME3 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLING.NODE,J,1) NE 0 OR
    INFO(CALLING.NODE,J,4) NE 0
  LET SPECINFO(CALLED.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
   LET SPECINFO(CALLED.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
LET FRAME3 = 0
FOR J = K TO 12 , DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0 AND
 INFO(CALLING.NODE,J,1) EQ 0
 LET SLOT3 = J
 GO TO CONT1
ALWAYS
LOOP
''
LET FRAME3 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLING.NODE,J,1) EQ 0 AND INFO(CALLING.NODE,J,4)
    EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0
  LET SLOT3 = J
 GO TO CONT1
ALWAYS
LOOP
GO TO YYYY
''
'NEXT.FRAME3'
''
LET FRAME3 = 1
FOR J = 1 TO 12, DO
 IF INFO(CALLING.NODE,J,1) NE 0 OR
    INFO(CALLING.NODE,J,4) NE 0
  LET SPECINFO(CALLED.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
 IF INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
 AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT3 = J
 GO TO CONT1
```

```
ALWAYS
LOOP
''
'YYYY'
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** FAILED IN EVENT RESPONSE TO REQUEST
SKIP 1 OUTPUT LINE
''
'UNSUCCESS'
''
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET CKT.FAILED = CKT.FAILED + 1
LET UP.ROUTE = UP.ROUTE - 1
LET DOWN.ROUTE = DOWN.ROUTE + 1
''
IF PRNT EQ 0
 PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AND TIME.V
                AS FOLLOWS
CIRCUIT **** FAILED TO CONNECT AT TIME ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET DIRECTION(MESSAGE) = 3
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE NOW
RETURN
''
'' CONT1 IDENTIFIES A  SLOT TO CARRY THE SERVICE MESSAGE
'' TO THE CALLED NODE AND ALSO COMPUTES WHEN THE SERVICE
'' MESSAGE WILL ARRIVE AT THE CALLED NODE
''
'CONT1'
''
LET DELAY3 = REAL.F(12 * FRAME3 + SLOT3 - CURRENT.SLOT)
               * SLOT.DURATION
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT3
LET SLOT.ASSIGN(MESSAGE) = 0
LET RECSLOT(MESSAGE) = 0
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE), FM.NODE(MESSAGE)
                    AND (TIME.V + DELAY3) AS FOLLOWS
CIRCUIT ***** HAS SCHEDULED A REQ FOR SERVICE AT NODE ** AT
TIME ***.*****
SKIP 2 OUTPUT LINES
ALWAYS
''
SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE AT TIME.V + DELAY3
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE), FM.NODE(MESSAGE),
          AND (TIME.V + DELAY3)  AS FOLLOWS
CIRCUIT **** HAS SCHEDULED A RESPONSE TO SVC
        AT NODE ** AT ***.**
SKIP 1 OUTPUT LINES
''
PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT END OF RESPONSE TO SVC ARE :
LIST ATTRIBUTES OF MESSAGE
SKIP 2 OUTPUT LINES
ALWAYS
''
RETURN
END  '' RESPONSE TO REQUEST
''
''
''
```

```
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT UPSTREAM.BREAK.DOWN GIVEN U.B.D.MSG
''
'' THIS EVENT BREAKS DOWN A ESTABLISHED CIRCUIT
'' FROM THE ORIGINATOR NODE TO THE DESTINATION NODE
''
'' IT REMOVES SLOT ASSIGNMENTS FROM THE NODAL SLOT
'' ASSIGNMENT TABLES SO THAT THESE RELEASED SLOTS
'' CAN BE USED IN THE ESTABLISHMENT OF OTHER CIRCUITS
''
'' THIS EVENT SELECTS A RELEVANT PORTION OF PROGRAM TO
'' EXECUTE DEPENDING ON THE VALUE OF DIRECTION(MESSAGE)
''
'' -2 : START BREAKING DOWN AN ESTABLISHED CIRCUIT FROM
''      THE ORIGINATOR NODE TO THE DESTINATION NODE
''
'' -1 : CONTINUE BREAKING DOWN AN ESTABLISHED CIRCUIT FROM
''      AN INTERMEDIATE NODE TO THE DESTINATION NODE
''
''  0 : BREAK DOWN WHEN A RESPONSE TO REQ FAILED
''
LET MESSAGE = U.B.D.MSG
''
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
UPSTREAM BREAK DOWN PERFORMED AT TIME ****.******
 SKIP 2 OUTPUT LINES
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF THE ENTITY AT START OF UPSTREAM BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF TYPE(MESSAGE) EQ 1
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
''
IF DIRECTION(MESSAGE) EQ -1
  GO TO CONT.BREAKDOWN
ALWAYS
''
 IF DIRECTION(MESSAGE) EQ 0
  GO TO RESPONSE.BREAKDOWN
ALWAYS
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ -2 AND
   TYPE(MESSAGE) EQ FULL.BREAKDOWN
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
 ORIGINATOR(MESSAGE),DESINATION(MESSAGE),
 TIME.V AND START.TIME(MESSAGE) AS FOLLOWS
 CIRCUIT ***** FM NODE ** TO NODE ** WAS ONCE ESTABLISHED
 BROKEN DOWN AT TIME ****.****** AFTER CARRYING VOICE
 TRAFFIC FOR A CALL DURATION OF ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
''
LET DOWN.ROUTE = DOWN.ROUTE + 1
LET DIRECTION(MESSAGE) = -1
''
FOR I = 1 TO 12 , DO
```

```
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT2.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4) =
      INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET SPECINFO(FM.NODE(MESSAGE),I) = 6
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  GO TO COMPUTE.DELAY
 ALWAYS
''
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN UPSTREAM BREAKDOWN FOR CIRCUIT NO. *****
SKIP 1 OUTPUT LINE
RETURN
''
''  WE HAVE SET THE TRANSMIT AND RECEIVE SLOTS AT THE
''  ORIGINATOR NODE TO ZERO.
''  WE NOW BREAK DOWN THE CIRCUIT ALONG THE UPSTREAM PATH
''  FROM THE ORIGINATOR NODE TO THE DESTINATION NODE.
''
''  CHECK WHETHER WE ARE AT THE DESTINATION NODE ,
''  IF SO ,WE NEED ONLY DELETE THE TRANSMIT AND RECEIVE
''  SLOT ASSIGNMENTS FOR THIS CIRCUIT AND
''  COLLECT STATISTICS DATA
''
'CONT.BREAKDOWN'
''
LET SLOT1.XMIT = RECSLOT(MESSAGE)
LET SLOT1.REC = INFO(TO.NODE(MESSAGE),SLOT1.XMIT,2)
LET INFO(TO.NODE(MESSAGE),SLOT1.XMIT,1) = 0
LET SPECINFO(TO.NODE(MESSAGE),SLOT1.XMIT) = 6
LET INFO(TO.NODE(MESSAGE),SLOT1.XMIT,2) = 0
LET INFO(TO.NODE(MESSAGE),SLOT1.XMIT,3) = 0
LET INFO(TO.NODE(MESSAGE),SLOT1.REC,4) =
''    INFO(TO.NODE(MESSAGE),SLOT1.REC,4) -1
''
''  WE HAVE COMPLETED RELEASING THE DOWN-SIDE
''  RECEIVE AND TRANSMIT SLOT ASSIGNMENTS
''
''  IF WE ARE AT THE DESTINATION NODE,WE CAN NOW COLLECT
''  STATISTICS, OTHERWISE,
''  WE WILL CONTINUE BREAKING DOWN THE UP-SIDE SLOT
''  ASSIGNMENTS
''
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
 RETURN
ALWAYS
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT2.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4) =
      INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),I) = 6
```

```
   LET INFO(FM.NODE(MESSAGE),I,2) = 0
   LET INFO(FM.NODE(MESSAGE),I,3) = 0
   LET DIRECTION(MESSAGE) = -1
   GO TO COMPUTE.DELAY
  ALWAYS
 LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** HAS FAULT IN EVENT UPSTREAM BREAK DOWN
SKIP 1 OUTPUT LINE
RETURN
''
''  USES THE ASSIGNED TRANSMIT SLOT TO CARRY THE BREADDOWN
''  MESSAGE TO THE NEXT NODE UPSTREAM
''  ON THE WAY TO THE DESTINATION NODE.
''
''  COMPUTE WHEN THE BREAK DOWN MESSAGE WILL ARRIVE AT
''  THE NEXT NODE
''
'COMPUTE.DELAY'
''
IF SLOT2.XMIT GT (CURRENT.SLOT + 1)
 LET DELAY = SLOT2.XMIT - CURRENT.SLOT
 GO TO NEXT.BREAKDOWN
ALWAYS
''
IF SLOT2.XMIT EQ (CURRENT.SLOT + 1)
 LET DELAY = 13
 GO TO NEXT.BREAKDOWN
ALWAYS
''
IF SLOT2.XMIT LT (CURRENT.SLOT + 1)
 LET DELAY = SLOT2.XMIT - CURRENT.SLOT + 12
 GO TO NEXT.BREAKDOWN
ALWAYS
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN UPSTREAM BD DELAY CALCULATION FOR CIRCUIT ****
SKIP 1 OUTPUT LINE
RETURN
''
'NEXT.BREAKDOWN'
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT2.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN MESSAGE
         AT TIME.V + INCREMENT
GO TO LAST.UPSTREAM
''
'RESPONSE.BREAKDOWN'
''
IF RECSLOT(MESSAGE) EQ 13
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
ALWAYS
''
IF RECSLOT(MESSAGE) LE 12
 DESTROY THE MESSAGE CALLED MESSAGE
ALWAYS
''
'LAST.UPSTREAM'
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF THE ENTITY AT END OF UPSTREAM BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 1 OUTPUT LINE
ALWAYS
''
```

```
''
RETURN
END '' UPSTREAM BREAK DOWN
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT DOWNSTREAM.BREAK.DOWN GIVEN D.B.D.MSG
''
'' THIS EVENT BREAKS DOWN A ESTABLISHED CIRCUIT
'' IN THE DOWNSTREAM DIRECTION,THAT IS,
'' FROM THE DESTINATION NODE TO THE ORIGINATOR
'' NODE. THE CIRCUIT CAN BE FULLY OR PARTIALLY ESTABLISHED
''
'' IT REMOVES SLOT ASSIGNMENTS FROM THE NODAL SLOT
'' ASSIGNMENT TABLES SO THAT THESE RELEASED SLOTS
'' CAN BE USED IN THE ESTABLISHMENT OF
'' OTHER CIRCUITS
''
'' THIS EVENT SELECTS A RELEVANT PORTION OF PROGRAM TO
'' EXECUTE DEPENDING ON THE VALUE OF DIRECTION(MESSAGE)
''
''  1 : START BREAKING DOWN AN ESTABLISHED CIRCUIT
''      FROM THE DESTINATION NODE TO THE ORIGINATOR NODE
''
''  2 : CONTINUE BREAKING DOWN AN ESTABLISHED CIRCUIT FROM
''      AN INTERMEDIATE NODE TO THE ORIGINATOR NODE
''
''  3 : START BREAKING DOWN FROM A NODE TO THE ORIGINATOR
''      NODE CALLED BY REQUEST FOR SERVICE
''  4 : START BREAKING DOWN FROM A NODE TO THE ORIGINATOR
''      NODE CALLED BY RESPONSE TO REQUEST
''
''
LET MESSAGE = D.B.D.MSG
''
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 DOWNSTREAM BREAK DOWN PERFORMED AT TIME ****.******
 SKIP 2 OUTPUT LINES
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF THE ENTITY AT START OF DOWNSTREAM BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF TYPE(MESSAGE) EQ 1
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
''
IF DIRECTION(MESSAGE) EQ  1
  GO TO ONE
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  2
  GO TO TWO
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  3
  GO TO THREE
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  4
```

```
  GO TO FOUR
ALWAYS
''
'ONE'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 1
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
 ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
 START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
 CIRCUIT **** FM ** TO  ** WAS ESTABLISHED FOR A CALL
 DURATION OF ****.******* SECS IS BEING BROKEN DOWN
 IN THE DOWNSTREAM AT TIME ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET FM.NODE(MESSAGE) = DESTINATION(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
LET DOWN.ROUTE = DOWN.ROUTE + 1
LET DIRECTION(MESSAGE) = 2
''
'JUMP.IN'
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT1.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET MM = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = MM
  LET INFO(FM.NODE(MESSAGE),MM,4) =
      INFO(FM.NODE(MESSAGE),MM,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),I) = 6
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  LET DIRECTION(MESSAGE) = 2
  GO TO COMPUTE.DELAY
 ALWAYS
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN EVENT DOWNSTREAM BD FOR CIRCUIT *****
SKIP 1 OUTPUT LINE
RETURN
''
'' WE HAVE SET THE TRANSMIT AND RECEIVE SLOTS AT
'' THE DESTINATION NODE TO ZERO
'' WE NOW BREAK DOWN THE CIRCUIT ALONG THE DOWNSTREAM PATH
''
'' CHECK WHETHER WE ARE AT THE ORIGINATOR NODE ,
'' IF SO ,WE NEED ONLY DELETE THE TRANSMIT AND
'' AND RECEIVE SLOT ASSIGNMENTS FOR THIS CIRCUIT
'' AND COLLECT STATISTICS DATA
''
'TWO'
''
LET SLOT2.XMIT = RECSLOT(MESSAGE)
LET SLOT2.REC = INFO(TO.NODE(MESSAGE),SLOT2.XMIT,2)
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,1) = 0
LET SPECINFO(TO.NODE(MESSAGE),SLOT2.XMIT) = 6
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,2) = 0
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,3) = 0
LET INFO(TO.NODE(MESSAGE),SLOT2.REC,4) =
    INFO(TO.NODE(MESSAGE),SLOT2.REC,4) - 1
''
'' WE HAVE COMPLETED RELEASING THE UP-SIDE RECEIVE AND
'' TRANSMIT SLOT ASSIGNMENTS
''
'' IF WE ARE AT THE ORIGINATOR NODE,WE CAN NOW COLLECT
'' STATISTICS, OTHERWISE
```

```
'' WE WILL CONTINUE BREAKING DOWN THE DOWN SIDE SLOT
'' ASSIGNMENTS
''
IF TO.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
 RETURN
ALWAYS
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT1.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4) =
      INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),I) = 6
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  LET DIRECTION(MESSAGE) = 2
  GO TO COMPUTE.DELAY
 ALWAYS
''
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** HAS FAULT IN EVENT DOWN BREAKDOWN
RETURN
''
'THREE'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 3
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
 ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
 START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
 CIRCUIT ***** FM ** TO  ** CANNOT BE ESTABLISHED.
 BEGIN TO BREAK DOWN THE CIRCUIT AT TIME ****.******
 TIME NOW IS ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET DIRECTION(MESSAGE) = 2
GO TO JUMP.IN
''
'FOUR'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 4
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
        ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
        START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
CIRCUIT ***** FM ** TO  ** CANNOT BE ESTABLISHED.
BEGIN TO BREAK DOWN THE CIRCUIT AT TIME ****.******
TIME NOW IS ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET DIRECTION(MESSAGE) = 2
GO TO JUMP.IN
''
'' USE THE ASSIGNED TRANSMIT SLOT TO CARRY
'' THE BREADDOWN MESSAGE TO THE NEXT NODE
'' UPSTREAM ON THE WAY TO THE DESTINATION NODE.
''
'' COMPUTE WHEN THE BD MSG WILL ARRIVE AT THE NEXT NODE
''
```

```
''COMPUTE.DELAY'

IF SLOT1.XMIT GT (CURRENT.SLOT + 1)
 LET DELAY = SLOT1.XMIT - CURRENT.SLOT
 GO TO LAST.DOWN
ALWAYS
''
IF SLOT1.XMIT EQ (CURRENT.SLOT + 1)
 LET DELAY = 13
 GO TO LAST.DOWN
ALWAYS
''
IF SLOT1.XMIT LT (CURRENT.SLOT + 1)
 LET DELAY = SLOT1.XMIT - CURRENT.SLOT + 12
 GO TO LAST.DOWN
ALWAYS
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
 FAULT IN DOWNSTREAM BD DELAY COMPUTATION AT CKT ****
RETURN
''
''LAST.DOWN'
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT1.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF THE ENTITY AT END OF DOWNSTREAM BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 1 OUTPUT LINE
ALWAYS
''
SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE AT TIME.V + INCREMEN
''
RETURN
END '' DOWNSTREAM BREAKDOWN
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR VIRTUAL.CKT GIVEN MSG
''
'' THIS ROUTE COLLECTS STATISTICS ON CIRCUITS THAT ARE
'' ESTABLISHED AND SCHEDULES THEIR EVENTUAL
'' DISESTABLISHMENT ACCORDING TO AN EXPONENTIAL
'' DISTRIBUTION FUNCTION WITH A MEAN CALL DURATION
''
LET MESSAGE = MSG
''
DEFINE CALL.END.TIME AS A REAL VARIABLES
''
IF PRNT EQ 1
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
ROUTINE VIRTUAL CIRCUIT PERFORMED AT TIME ****.******
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY WHEN VIRTUAL CKT WAS CALLED ARE :
 LIST ATTRIBUTE OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET CKT.ESTAB = CKT.ESTAB + 1
LET UP.ROUTE = UP.ROUTE - 1
```

```
LET DELAY.SUM = DELAY.SUM + START.TIME(MESSAGE)
LET AVG.TIME.EST = DELAY.SUM /REAL.F(CKT.ESTAB)
''
'' DID THIS CIRCUIT TAKE THE MOST TIME TO ESTABLISH
''
IF START.TIME(MESSAGE) GT LONG.TIME.EST
 LET LONG.TIME.EST = START.TIME(MESSAGE)
 LET CKT.LONG.TIME.EST = CKT.NUMBER(MESSAGE)
ALWAYS
''
'' SCHEDULES THE TIME FOR THE NEWLY ESTABLISHED CIRCUIT
'' TO BE ACTIVE AND SELECTS FROM EITHER ORIGINATOR
'' NODE OF DESTINATION THE CIRCUIT TO BE
'' DISESTABLISHED AND SCHEDULES THE EVENT TO BREAK DOWN
'' THE CIRCUIT
''
LET CALL.DURATION = EXPONENTIAL.F(MEAN.CALL.DURATION,7)
LET CALL.END.TIME = CALL.DURATION + TIME.V
LET SUM.DURATION = SUM.DURATION + CALL.DURATION
LET AVG.DURATION = SUM.DURATION / REAL.F(CKT.ESTAB)
LET START.TIME(MESSAGE) = CALL.DURATION
LET TYPE(MESSAGE) = FULL.BREAKDOWN
''
LET SLOT.ARRIVAL(MESSAGE) = RANDI.F(1,12,4)
IF PRNT EQ 0
 PRINT 1 LINE WITH SLOT.ARRIVAL(MESSAGE) AS FOLLOWS
CIRCUIT BEGIN BREAKING DOWN IN SLOT **
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF FAIR.POINTER EQ 1
 LET FAIR.POINTER = 0
 LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
 LET DIRECTION(MESSAGE) = -2
 SCHEDULE AN UPSTREAM.BREAK.DOWN GIVEN MESSAGE
          AT CALL.END.TIME
 GO TO LAST.VIRTUAL
ALWAYS
''
IF FAIR.POINTER EQ 0
 LET FAIR.POINTER = 1
 LET DIRECTION(MESSAGE) = 1
 SCHEDULE A DOWNSTREAM.BREAK.DOWN GIVEN MESSAGE
          AT CALL.END.TIME
ALWAYS
''
'LAST.VIRTUAL'
''
IF PRNT EQ 0 AND FAIR.POINTER EQ 0
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
 ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
 TIME.V, CALL.DURATION AND CALL.END.TIME
              AS FOLLOWS
CIRCUIT  **** FM  ** TO  ** WAS ESTABLISHED AT TIME
***.***** AND HAS CALL DURATION OF ****.****** SECS,
BREAKDOWN BEGIN IN THE UPSTREAM DIRECTION AT ****.******
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 0 AND FAIR.POINTER EQ 1
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
 ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
 TIME.V, CALL.DURATION AND CALL.END.TIME
              AS FOLLOWS
CIRCUIT ***** FM ** TO ** WAS ESTABLISHED AT TIME
***.***** AND HAS CALL DURATION OF ****.****** SECS,
BREAKDOWN BEGINS IN THE DOWNSTREAM DIRECTION AT ****.******
 SKIP 1 OUTPUT LINE
ALWAYS
```

```
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT THE END OF VIRTUAL.CKT ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
RETURN
END  '' VIRTUAL CKT
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR STATS.AT.END.BREAK.DOWN GIVEN B.D.MESSAGE
''
'' THIS ROUTINE COLLECTS STATISTICS OF THE CIRCUIT
'' THAT ARE BROKEN DOWN
''
LET MESSAGE = B.D.MESSAGE
''
DEFINE BD.TIME AS A REAL VARIABLE
''
IF TYPE(MESSAGE) EQ FULL.BREAKDOWN
 LET CKT.DISESTAB = CKT.DISESTAB + 1
ALWAYS
''
LET CKTS.BD = CKT.DISESTAB + CKT.FAILED
LET DOWN.ROUTE = DOWN.ROUTE - 1
LET BD.TIME = START.TIME(MESSAGE)
LET SUM.BD.TIME = SUM.BD.TIME + BD.TIME
LET AVG.BD.TIME = SUM.BD.TIME / REAL.F(CKTS.BD)
''
''
'' COLLECTS STATS ON THE BREAK DOWN OF
'' PARTIALLY ESTABLISHED CIRCUITS
''
''
IF TYPE(MESSAGE) EQ PARTIAL.BREAKDOWN
 IF START.TIME(MESSAGE) GT LONG.P.BD
  LET LONG.P.BD = START.TIME(MESSAGE)
 ALWAYS
''
 LET TOT.P.BD = TOT.P.BD + START.TIME(MESSAGE)
 LET P.BD.COUNTER = P.BD.COUNTER + 1
 LET AVG.P.BD = TOT.P.BD / REAL.F(P.BD.COUNTER)
ALWAYS
''
'' COLLECTS STATS ON THE BREAK DOWN OF
'' FULLY ESTABLISHED CIRCUITS
''
IF TYPE(MESSAGE) EQ FULL.BREAKDOWN
 IF START.TIME(MESSAGE) GT LONG.C.BD
  LET LONG.C.BD = START.TIME(MESSAGE)
 ALWAYS
''
 LET TOT.C.BD = TOT.C.BD + START.TIME(MESSAGE)
 LET C.BD.COUNTER = C.BD.COUNTER + 1
 LET AVG.C.BD = TOT.C.BD / REAL.F(C.BD.COUNTER)
ALWAYS
''
DESTROY THE MESSAGE CALLED MESSAGE
''
RETURN
''
END '' STATS AT END BREAKDOWN
''
```

```
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
EVENT HALT.SIMULATION SAVING THE EVENT NOTICE
''
''
'' THIS ROUTINES HALTS THE PROGRAM AND PRINTS
'' ANALYSIS STATMENTS AT THE END OF A SIMULATION RUN
''
LET PRNT.COUNTER = PRNT.COUNTER + 1
''
START NEW PAGE
PRINT 1 LINE WITH PRNT.COUNTER AS FOLLOWS
THIS IS THE ** TH SIMULATION RUN
SKIP 1 OUTPUT LINE
''
''
IF (CKT.GENERATED - UP.ROUTE) GT 0
 LET FRACT.SUCCESSFUL.CALL = REAL.F(CKT.ESTAB) * 100.00 /
                        REAL.F(CKT.GENERATED - UP.ROUTE)
 LET FRACT.LOST.CALL = 100.00 - FRACT.SUCCESSFUL.CALL
ALWAYS
''
PRINT 16 LINES WITH  CKT.SUM, CKT.GENERATED, CKT.ESTAB,
         CKT.DISTAB, CKT.FAILED, OFFERED.TRAFFIC,
         AVG.TIME.EST, LONG.TIME.EST,CKT.LONG.TIME.EST,
         AVG.DURATION, P.BD.COUNTER,
         C.BD.COUNTER, AVG.C.BD, SLOT.DEPTH,
         FRACT.SUCCESSFUL.CALL AND FRACT.LOST.CALL
               AS FOLLOWS
STATISTICS OF THIS SIMULATION :
THE NUMBER OF CIRCUIT CREATED SO FAR = *****
THE NUMBER OF CIRCUIT GENERATED = *****
THE NUMBER OF CIRCUIT ESTABLISHED = ****
THE NR OF ESTABLISHED CKTS THAT ARE DISESTABLISHED ****
THE NUMBER OF CIRCUITS WERE NOT ESTABLISHED = ****
THE OFFERED TRAFFIC IS **
THE AVERAGE TIME TO ESTABLISH A CIRCUIT = ***.******
THE LONGEST TIME TO ESTABLISH A CIRCUIT  ***.**** AT ****
THE AVERAGE DURATION OF AN ESTABLISHED CIRCUITS ***.*****
THE NUMBER OF PARTIALLY ESTABLISHED CIRCUITS = ****
THE NUMBER OF FULLY ESTABLISHED CIRCUITS = ****
THE AVERAGE TIME TO BREAK DOWN A COMPLETED CIRCUIT ***.***
THE SLOT DEPTH IS **
PERCENTAGES OF SUCCESSFULL CALL = ***.*****
PERCENTAGES OF LOST CALL = ***.******
SKIP 3 OUTPUT LINES
''
FOR NODE = 1 TO 2 , DO
''
  LET EMPTY = 0
  LET TRANSMIT.SLOTS = 0
  LET RECEIVE.SIGS = 0
  LET RECEIVE.SLOTS = 0
''
  RESERVE SLOTS.PER.FRAME(*) AS 12
  FOR S = 1 TO 12 , DO
   IF INFO(NODE,S,4) GE 1
    LET RECEIVE.SIGS = RECEIVE.SIGS + INFO(NODE,S,4)
    LET RECEIVE.SLOTS = RECEIVE.SLOTS + 1
    LET SLOTS.PER.FRAME(S) = INFO(NODE,S,4)
    GO TO OUT
   ALWAYS
''
   IF INFO(NODE,S,1) GT 0
    LET TRANSMIT.SLOTS = TRANSMIT.SLOTS + 1
    LET SLOTS.PER.FRAME(S) = 10000 + INFO(NODE,S,3)
```

```
    GO TO OUT
''  ALWAYS

    IF INFO(NODE,S,1) EQ 0 AND INFO(NODE,S,4) EQ 0
     LET EMPTY = EMPTY + 1
     LET SLOTS.PER.FRAME(S) = 0
''  ALWAYS

'OUT'
''
''  LOOP

   PRINT 2 LINES WITH NODE, EMPTY, TRANSMIT.SLOTS,
         RECEIVE.SIGS AND RECEIVE.SLOTS AS FOLLOWS
   NODE ** HAS ** EMPTY SLOTS, ** TRANSMIT SLOTS, AND
   HAS ** RECEIVE SIGNAL STACKED IN ** RECEIVE SLOTS
'' SKIP 2 OUTPUT LINES
''
''  PRINT THE TIME SLOT ASSIGNMENT AT EACH NODE
''
 PRINT 1 LINE WITH SLOTS.PER.FRAME(1),SLOTS.PER.FRAME(2),
             SLOTS.PER.FRAME(3),SLOTS.PER.FRAME(4),
             SLOTS.PER.FRAME(5),SLOTS.PER.FRAME(6),
             SLOTS.PER.FRAME(7),SLOTS.PER.FRAME(8),
             SLOTS.PER.FRAME(9),SLOTS.PER.FRAME(10),
             SLOTS.PER.FRAME(11) AND SLOTS.PER.FRAME(12)
                        AS FOLLOWS
**** **** **** **** **** **** **** **** **** **** **** ****
'' SKIP 2 OUTPUT LINES

   RELEASE SLOTS.PER.FRAME(*)
LOOP
''
PERFORM TERMINATION
''
RETURN
END '' HALT.SIMULATION
''
''
''
''  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR TERMINATION
''
FOR EACH NEW.CALL IN EV.S(I.NEW.CALL) , DO
 CANCEL THE NEW.CALL
 DESTROY THE NEW.CALL
LOOP
''
FOR EACH REQUEST.FOR.SVC IN EV.S(I.REQUEST.FOR.SVC),DO
 CANCEL THE REQUEST.FOR.SVC
 DESTROY THE REQUEST.FOR.SVC
LOOP
''
FOR EACH RESPONSE.TO.REQUEST
IN EV.S(I.RESPONSE.TO.REQUEST),DO
 CANCEL THE RESPONSE.TO.REQUEST
 DESTROY THE RESPONSE.TO.REQUEST
LOOP
''
FOR EACH UPSTREAM.BREAK.DOWN
IN EV.S(I.UPSTREAM.BREAK.DOWN), DO
 CANCEL THE UPSTREAM.BREAK.DOWN
 DESTROY THE UPSTREAM.BREAK.DOWN
LOOP
''
```

```
FOR EACH DOWNSTREAM.BREAK.DOWN
IN EV.S(I.DOWNSTREAM.BREAK.DOWN), DO
 CANCEL THE DOWNSTREAM.BREAK.DOWN
 DESTROY THE DOWNSTREAM.BREAK.DOWN
LOOP
''
FOR EACH HALT.SIMULATION
IN EV.S(I.HALT.SIMULATION), DO
 CANCEL THE HALT.SIMULATION
 DESTROY THE HALT.SIMULATION
LOOP
''
''
RETURN
END '' TERMINATION
/*
```

APPENDIX E

SIMULATION PROGRAM FOR EVALUATING STATIC AND DYNAMIC CONTROL

```
//DIJK   JOB (3060,0203),'FLOW',CLASS=J
//*MAIN ORG=NPGVM1.3060P,LINES=(20)
//  EXEC SIM25CLG,REGION.GO=4096K,PARM.GO='MAP,SIZE=760K'
//SIM.SYSIN DD *
''
PREAMBLE
''
NORMALLY MODE IS INTEGER
''
GENERATE LIST ROUTINES
''
TEMPORARY ENTITIES......
 EVERY MESSAGE HAS A CKT.NUMBER, A TYPE, AN ORIGINATOR,
  A DESTINATION, A FM.NODE, A TO.NODE,
  A START.TIME, A SLOT.ARRIVAL, A SLOT.ASSIGN,
''A RECSLOT, A DIRECTION, A REATTEMPT
''
 DEFINE START.TIME AS A REAL VARIABLE
''
''
EVENT NOTICES INCLUDE REQUEST.FOR.SVC,RESPONSE.TO.REQUEST,
      TO.DEST.BREAKDOWN, TO.ORG.BREAKDOWN,
''    NEW.CALL, DIJK.MANIPULATION AND HALT.SIMULATION

      EVERY REQUEST.FOR.SVC HAS A MSG1
      EVERY RESPONSE.TO.REQUEST HAS A MSG2
      EVERY TO.DEST.BREAKDOWN HAS A BDTODEST
      EVERY TO.ORG.BREAKDOWN HAS A BDTOORG
''
''
PRIORITY ORDER IS NEW.CALL, TO.DEST.BREAKDOWN,
            TO.ORG.BREAKDOWN,REQUEST.FOR.SVC,
        RESPONSE.TO.REQUEST,DIJK.MANIPULATION
                         AND HALT.SIMULATION
''
''
DEFINE INFO AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE SPECINFO AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE SLOTS.PER.FRAME AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE ATTENUATION AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE LINKCONNECT AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE MEANY.GIVEN AS A 1-DIMENSIONAL REAL ARRAY
DEFINE PROBY.GIVEN AS A 1-DIMENSIONAL REAL ARRAY
DEFINE SLOT.DEPTH AND N AS INTEGER VARIABLES
DEFINE DIJKSTRA AND DISTANCE AS 2-DIMENSIONAL REAL ARRAYS
DEFINE LINK.ATTENU AS A 2-DIMENSIONAL REAL ARRAY
DEFINE UP.DATE.PERIOD AS A REAL VARIABLE
DEFINE HOP.GREATEST,HOP.SUM AND HOP.AVG AS REAL VARIABLES
DEFINE C.LEVEL AS A 2-DIMENSIONAL REAL ARRAY
DEFINE AVAILCHANNEL AS A 2-DIMENSIONAL REAL ARRAY
DEFINE TOT.HOP.GREATEST AS A REAL VARIABLE
DEFINE N.LINKS AND MAX.LINKS.PER.NODE AS INTEGER VARIABLES
DEFINE CALLED.NODE AND CALLING.NODE AS INTEGER VARIABLES
DEFINE LINKS AND LINK.NODE.RATIO AS REAL VARIABLES
DEFINE NODETRAFFIC AS A REAL VARIABLE
DEFINE BEST.PATH AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE MEANY AS A 2-DIMENSIONAL REAL ARRAY
DEFINE PRNT.COUNTER AS AN INTEGER VARIABLE
```

```
DEFINE PATHPRNT AS AN INTEGER VARIABLE
DEFINE CKT.ESTAB, CKT.FAILED, CKT.SUM AND CKT.DISESTAB AS
       INTEGER VARIABLES
DEFINE BK.TO.DEST AND BK.TO.ORG AS INTEGER VARIABLES
DEFINE PRNT AS AN INTEGER VARIABLE
DEFINE TEST.DURATION, SLOT.DURATION, MEAN.SYS.CALL.ARRIV,
      AND MEAN.CALL.DURATION AS REAL VARIABLES
DEFINE FAIR.POINTER AS AN INTEGER VARIABLE
DEFINE LONG.TIME.EST, AVG.P.BD, LONG.P.BD, AVG.C.BD,
      LONG.C.BD AND AVG.TIME.EST AS REAL VARIABLES
DEFINE CKT.LONG.TIME.EST AS AN INTEGER VARIABLE
DEFINE MAX.CKT AS AN INTEGER VARIABLE
DEFINE SUM.BD.TIME, AVG.BD.TIME, TOT.P.BD AND TOT.C.BD
       AS REAL VARIABLES
DEFINE CKTS.BD AS AN INTEGER VARIABLE
DEFINE FRACT.LOST.CALL AND FRACT.SUCCESSFUL.CALL
       AS  REAL VARIABLES
DEFINE C.BD.COUNTER AND P.BD.COUNTER AS INTEGER VARIABLES
DEFINE PATH.CONNECT AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE ORG.NODE AND DEST.NODE AS INTEGER VARIABLES
DEFINE SUM.DURATION AND CALL.DURATION AS REAL VARIABLES
DEFINE BREAKTIME AS A REAL VARIABLE
DEFINE NLINK.FOR.NODE AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE DELAY.SUM AND AVG.DURATION AS REAL VARIABLES
DEFINE OFFERED.TRAFFIC AS A REAL VARIABLE
DEFINE MAX.ATTEMPT AS AN INTEGER VARIABLE
DEFINE ALPHA AS AN INTEGER VARIABLE
DEFINE BEGIN.DIJK AS A REAL VARIABLE
DEFINE PARTIAL.BREAKDOWN TO MEAN 3
DEFINE FULL.BREAKDOWN TO MEAN 4
DEFINE DYNAMIC.ALGORITHM AS AN INTEGER VARIABLE
DEFINE CLEAN AS 1-DIMENSIONAL INTEGER ARRAY
DEFINE TOT.DIJK.CALLED AS AN INTEGER VARIABLE
DEFINE NSLOT.AVAIL.I AS AN INTEGER VARIABLE
''
END ''PREAMBLE
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
MAIN
''
LET LINE.V = 80
START NEW PAGE
''
PRINT 3 LINES AS FOLLOWS
THE OBJECTIVE OF THIS SIMULATION IS TO DETERMINE
THE EFFECTIVENESS OF THE PROPOSED DISTANCE FUNCTION
AND THE PROPOSED TIME SLOT ASSIGNMENT ALGORITHM.
SKIP 2 OUTPUT LINES
''
'' THE MAIN PROGRAM CALLS THE FRESH.INPUT ROUTINE THAT
'' SETS THE INPUT PARAMETERS AND INTIALIZATION VARIABLES
'' FOR SIMULATION
''
LET PRNT.COUNTER = 0
LET FAIR.POINTER = 1
LET TIME.V = 0.000000000
''
PERFORM FRESH.INPUT
''
RESERVE INFO(*,*,*) AS N BY 12 BY 4
RESERVE SPECINFO(*,*,*) AS N BY N BY 12
RELEASE SEED.V(*)
RESERVE SEED.V(*) AS 10
''
LET SEED.V(1) = 2116429302
```

```
LET SEED.V(2) =  683743814
LET SEED.V(3) =  964393174
LET SEED.V(4) = 1217426631
LET SEED.V(5) =  618433579
LET SEED.V(6) = 1157240309
LET SEED.V(7) =   15726055
LET SEED.V(8) =   48108509
LET SEED.V(9) = 1797920909
LET SEED.V(10) = 477424540
''
''
''  INFO(NODE,SLOT,INDEX) = INTEGER VALUE
''
''    NODE DENOTES NODE NUMBER
''    SLOT DENOTES SLOT NUMBER
''    INDEX = 0  : EMPTY SLOT
''    INDEX = 1  : TRANSMIT"S SLOT WITH CIRCUIT NUMBER
''    INDEX = 2  : RECEIVE"S SLOT FOR RETURN SIGNAL
''    INDEX = 3  : CALLED OR CALLING NODE NUMBER
''    INDEX = 4  : NUMBER OF RECEIVE SIGNALS
''
LET BEGIN.DIJK = 200.00
IF DYNAMIC.ALGORITHM EQ 1
   PERFORM DISTANCE.INITIALIZATION
   SCHEDULE A DIJK.MANIPULATION AT BEGIN.DIJK
ALWAYS
''
SCHEDULE A NEW.CALL AT
EXPONENTIAL.F(MEAN.SYS.CALL.ARRIV,5)
SCHEDULE A HALT.SIMULATION AT TEST.DURATION
''
START SIMULATION
''
SKIP 2 OUTPUT LINES
PRINT 1 LINE AS FOLLOWS
END OF SIMULATION
''
STOP
END
''
''
''
''  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR FRESH.INPUT
''
''  {                                                   }
''  {                                                   }
''  {      SAME ROUTINE AS IN APPENDIX F                }
''  {                                                   }
''  {                                                   }
''
''
''  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR DISTANCE.INITIALIZATION
''
''  THE DIJKSTRA ARRAY HOLDS A REAL POSITIVE NUMBER
''  INDICATING THE TOTAL OVERALL LINK "DISTANCE" FROM
''  EACH NODE TO EVERY OTHER NODE IN THE NETWORK.
''  INITIALLY, IF A DIRECT LINK EXISTS BETWEEN TWO NODES
''  WE SHALL ASSIGN A VALUE OF 1.0 AND IF A DIRECT LINK
''  DOES NOT EXIST, WE SHALL ASSIGN A VALUE OF 9999.0
''  THE VALUES IN THIS ARRAY WILL CHANGE DURING THE
''  SIMULATION AS INIDIVIDUAL LINK WEIGHTS CHANGE TO
''  REFLECT VARYING DEGREES OF LINK, NODE AND
''  NETWORK LOADING.
''
''
```

```
RESERVE DIJKSTRA(*,*) AS N BY N
RESERVE DISTANCE(*,*) AS N BY N
''
FOR I = 1 TO N, DO
 FOR J = 1 TO N, DO
  IF I EQ J
   LET DIJKSTRA(I,J) = 9999.0
   LET DISTANCE(I,J) = 9999.0
''ALWAYS

  IF I NE J
   LET DIJKSTRA(I,J) = 800.00
   LET DISTANCE(I,J) = 800.00
''ALWAYS

  IF LINKCONNECT(I,J) EQ 1
   LET DIJKSTRA(I,J) = 1.0
   LET DISTANCE(I,J) = 1.0
  ALWAYS
 LOOP
LOOP
''
'' PRINT ONE OF THESE ARRAYS TO ENSURE THEY WERE
'' SET UP PROPERLY.
''
 PRINT 4 LINES AS FOLLOWS
 THE CONTENTS OF THE INITIAL DISTANCE MATRIX ARE:
  +TO +    1      2       3       4        5        6        7
 FM + -----+---------------------------------------------------------
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2),
        DISTANCE(I,3),DISTANCE(I,4),DISTANCE(I,5),
        DISTANCE(I,6), DISTANCE(I,7)
              AS FOLLOWS
  **   + ****.* ****.* ****.* ****.* ****.* ****.* ****.*
 LOOP
 SKIP 1 OUTPUT LINE
 PRINT 5 LINES AS FOLLOWS
 CONTENTS OF THE INITIAL DISTANCE MATRIX (CONT.) :
  + TO +      8          9          10         11
 FM +   ------+--------------------------------------
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9),
        DISTANCE(I,10),DISTANCE(I,11) AS FOLLOWS
  **    + ******.* ******.*  ******.*  ******.*
 LOOP
 SKIP 2 OUTPUT LINES
''
RETURN
END '' DISTANCE INITIALIZATION
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
EVENT DIJK.MANIPULATION SAVING THE EVENT NOTICE
''
DEFINE DIST AS  A REAL VARIABLE
''
LET TOT.DIJK.CALLED = TOT.DIJK.CALLED + 1
''
PRINT 1 LINE WITH TOT.DIJK.CALLED AND TIME.V AS FOLLOWS
**** TH ROUTING MANIPULATION INVOKED AT ****.** SECONDS
SKIP 1 OUTPUT LINE
''
'' GET THE CURRENT LINK "WEIGHTS" OR "DISTANCE"
'' AT EVERY NODE AND ON ALL LINKS
'' OF THE NETWORK
''
```

```
PERFORM COMPUTE.CURRENT.DISTANCES
''
RESERVE PATH.CONNECT(*,*) AS N BY N
''
'' USE THE CURRENT NODE AND LINK WEIGHT INFORMATION IN THE
'' IMPLEMENTATION OF THE DIJKSTRA ALGORITHM .
''
'' START BY INITIALIZING THE DIJKSTRA ARRAYS.
''
'' IF THERE IS NO LINK WHICH DIRECTLY CONNECTS TWO NODES,
'' THEN THE LINK WEIGHT IS SET EQUAL TO 9999.0
''
'' WE ALSO READ A COPY OF THE PATH.CONNECT ARRAY
'' WHICH WILL BE USED DURING THE DIJK.MANIPULATION EVENT
''
''
FOR I = 1 TO N, DO
 FOR J = 1 TO N, DO
  IF I EQ J
   LET DIJKSTRA(I,J) = 9999.0
   LET BEST.PATH(I,J) = 0
   LET PATH.CONNECT(I,J) = 0
  ALWAYS
  IF I NE J
   LET DIJKSTRA(I,J) = 800.00
   LET BEST.PATH(I,J) = 0
   LET PATH.CONNECT(I,J) = 0
''ALWAYS
  IF LINKCONNECT(I,J) EQ 1
   LET DIJKSTRA(I,J) = DISTANCE(I,J)
   LET BEST.PATH(I,J) = J
   LET PATH.CONNECT(I,J) = 1
''ALWAYS
  LOOP
''LOOP
LET MANIP.COUNTER = 0
LET PASS.COUNTER = 0
''
'MORE.RUN'
''
LET AGAIN.FLAG = 0
LET PASS.COUNTER = PASS.COUNTER + 1
FOR ROW = 1 TO N, DO
 FOR COL = 1 TO N, DO
  IF ROW EQ COL
    GO TO NEXT.COL
  ELSE
  FOR TCOL = 1 TO N, DO
   IF TCOL EQ ROW
     GO TO NEXT.TCOL
   ELSE
   IF TCOL EQ COL
    GO TO NEXT.TCOL
   ELSE
    LET DIST = 0.0
    IF LINKCONNECT(ROW,TCOL) EQ 1
      LET DIST = DIJKSTRA(ROW,TCOL)
      IF PATH.CONNECT(TCOL,COL) EQ 1
       LET DIST = DIST + DIJKSTRA(TCOL,COL)
       IF DIST LT DIJKSTRA(ROW,COL)
         LET DIJKSTRA(ROW,COL) = DIST
         LET BEST.PATH(ROW,COL) = BEST.PATH(ROW,TCOL)
         LET PATH.CONNECT(ROW,COL) = 1
         LET AGAIN.FLAG = 1
         LET MANIP.COUNTER = MANIP.COUNTER + 1
       ALWAYS
```

```
      ALWAYS
    ALWAYS
''
'NEXT.TCOL'
''
    LOOP
'NEXT.COL'
   LOOP
 LOOP
''
IF AGAIN.FLAG EQ 1
  GO TO MORE.RUN
ALWAYS
''
''  PRINT THE MANIPULATED DIJKSTRA AND BEST.PATH MATRICES
''
''
IF PATHPRNT EQ 1
PRINT 2 LINES WITH PASS.COUNTER AND MANIP.COUNTER
      AS FOLLOWS
PASSES THROUGH THE DIJKSTRA ARRAY **** TIMES AND PERFORMS
A TOTAL OF *** MANIPULATIONS FOR BEST PATH ASSIGNMENTS
SKIP 1 OUTPUT LINE
''
 PRINT 4 LINES AS FOLLOWS
 THE CONTENTS OF THE  MANIPULATED DIJKSTRA MATRIX ARE:
  +TO +    1       2       3       4       5       6       7
 FM + -----+------------------------------------------------------
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DIJKSTRA(I,1), DIJKSTRA(I,2),
        DIJKSTRA(I,3), DIJKSTRA(I,4),DIJKSTRA(I,5),
        DIJKSTRA(I,6) AND DIJKSTRA(I,7) AS FOLLOWS
        FOLLOWS
  **   + ****.* ****.* ****.* ****.* ****.* ****.* ****.*
 LOOP
 SKIP 1 OUTPUT LINE
 PRINT 5 LINES AS FOLLOWS
 CONTENTS OF THE MANIPULATED DIJKSTRA MATRIX (CONT.) :
 +          + TO +      8          9          10         11
 FM  +      -------+--------------------------------------------
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DIJKSTRA(I,8), DIJKSTRA(I,9), DIJKSTRA(I,10),
        DIJKSTRA(I,11) AS FOLLOWS
  **    +  *****.*  *****.*   *****.*   *****.*
 LOOP
 SKIP 2 OUTPUT LINES
''
''
 PRINT 5 LINES AS FOLLOWS
 THE CONTENTS OF THE MANIPULATED BEST.PATH MATRIX ARE:
 + TO  +  1   2   3   4   5   6   7   8   9  10  11
   +     FM  + -----+------------------------------------------
 FOR I = 1 TO N, DO
 PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2),
         BEST.PATH(I,4), BEST.PATH(I,5), BEST.PATH(I,6),
         BEST.PATH(I,7), BEST.PATH(I,8), BEST.PATH(I,9),
         BEST.PATH(I,10), BEST.PATH(I,11)
              AS FOLLOWS
 **    +  **  **  **  **  **  **  **  **  **  **  **
 LOOP
 SKIP 2 OUTPUT LINES
ALWAYS
''
SCHEDULE A DIJK.MANIPULATION AT TIME.V + UP.DATE.PERIOD
''
RETURN
END '' DIJK.MANIPULATION
''
''
```

```
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
ROUTINE TO COMPUTE.CURRENT.DISTANCES
''  {                                        }
''  {                                        }
''  {     SAME ROUTINE AS IN APPENDIX F      }
''  {                                        }
''  {                                        }
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
ROUTINE FOR COMBINATION GIVEN TOP AND BOTTOM YIELDING ANS
''  {                                        }
''  {                                        }
''  {     SAME ROUTINE AS IN APPENDIX F      }
''  {                                        }
''  {                                        }
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
ROUTINE FOR FACTORIAL GIVEN IVALUE YIELDING FAC.VALUE
''  {                                        }
''  {                                        }
''  {     SAME ROUTINE AS IN APPENDIX F      }
''  {                                        }
''  {                                        }
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
EVENT NEW.CALL SAVING THE EVENT NOTICE
''  {                                      }
''  {                                      }
''  {     SAME event AS IN APPENDIX F      }
''  {                                      }
''  {                                      }
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
EVENT TO.DEST.BREAKDOWN GIVEN BDTODEST
''  {                                    }
''  {                                    }
''  {     SAME EVENT AS IN APPENDIX F    }
''  {                                    }
''  {                                    }
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
EVENT TO.ORG.BREAKDOWN GIVEN BDTOORG
''  {                                    }
''  {                                    }
''  {     SAME EVENT AS IN APPENDIX F    }
''  {                                    }
''  {                                    }
''
```

```
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR VIRTUAL.CKT GIVEN ESTABLISH.MSG
''
''  {                                      }
''  {                                      }
''  {    SAME ROUTINE AS IN APPENDIX F     }
''  {                                      }
''  {                                      }
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR STATS.AT.END.BREAK.DOWN GIVEN B.D.MESSAGE
''
''
''  {                                      }
''  {                                      }
''  {    SAME ROUTINE AS IN APPENDIX F     }
''  {                                      }
''  {                                      }
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
EVENT HALT.SIMULATION SAVING THE EVENT NOTICE
''
''  {                                      }
''  {                                      }
''  {    SAME EVENT AS IN APPENDIX F       }
''  {                                      }
''  {                                      }
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR TERMINATION
''
FOR EACH NEW.CALL IN EV.S(I.NEW.CALL), DO
 CANCEL THE NEW.CALL
 DESTROY THE NEW.CALL
LOOP
''
FOR EACH REQUEST.FOR.SVC IN EV.S(I.REQUEST.FOR.SVC), DO
 CANCEL THE REQUEST.FOR.SVC
 DESTROY THE REQUEST.FOR.SVC
LOOP
''
FOR EACH RESPONSE.TO.REQUEST
IN EV.S(I.RESPONSE.TO.REQUEST), DO
 CANCEL THE RESPONSE.TO.REQUEST
 DESTROY THE RESPONSE.TO.REQUEST
LOOP
''
FOR EACH TO.DEST.BREAKDOWN
IN EV.S(I.TO.DEST.BREAKDOWN), DO
 CANCEL THE TO.DEST.BREAKDOWN
 DESTROY THE TO.DEST.BREAKDOWN
LOOP
''
FOR EACH TO.ORG.BREAKDOWN
IN EV.S(I.TO.ORG.BREAKDOWN), DO
 CANCEL THE TO.ORG.BREAKDOWN
 DESTROY THE TO.ORG.BREAKDOWN
LOOP
''
FOR EACH DIJK.MANIPULATION
IN EV.S(I.DIJK.MANIPULATION), DO
```

```
 CANCEL THE DIJK.MANIPULATION
 DESTROY THE DIJK.MANIPULATION
LOOP

FOR EACH HALT.SIMULATION IN EV.S(I.HALT.SIMULATION), DO
 CANCEL THE HALT.SIMULATION
 DESTROY THE HALT.SIMULATION
LOOP

RETURN
END '' TERMINATION
/*
//GO.SYSIN  DD *
 {  SAME LINK MATRIX AS IN APPENDIX F  }
/*
```

APPENDIX F

SIMULATION PROGRAM FOR EVALUATING YEN ROUTING CONDITIONS

```
//YEN152 JOB (3060,0203),'RICHNET',CLASS=J
//*MAIN ORG=NPGVM1.3060P,SYSTEM=SY1,LINES=(25)
//  EXEC SIM25CLG,REGION.GO=4096K,PARM.GO='MAP,SIZE=760K'
//SIM.SYSIN DD *
''
''
PREAMBLE
''
NORMALLY MODE IS INTEGER
''
GENERATE LIST ROUTINES
''
TEMPORARY ENTITIES.......
 EVERY MESSAGE HAS A CKT.NUMBER, A TYPE, AN ORIGINATOR,
       A DESTINATION, A FM.NODE, A TO.NODE,
       A START.TIME, A SLOT.ARRIVAL, A SLOT.ASSIGN,
''     A RECSLOT, A DIRECTION, A REATTEMPT
 DEFINE START.TIME AS A REAL VARIABLE
''
''
EVENT NOTICES INCLUDE REQUEST.FOR.SVC,RESPONSE.TO.REQUEST,
      TO.DEST.BREAKDOWN, TO.ORG.BREAKDOWN, SLOT.FOR.YEN,
''    NSLOT.FOR.YEN, NEW.CALL, YEN.ROUTING AND HALT.SIMULATION

      EVERY REQUEST.FOR.SVC HAS A MSG1
      EVERY RESPONSE.TO.REQUEST HAS A MSG2
      EVERY TO.DEST.BREAKDOWN HAS A BDTODEST
      EVERY TO.ORG.BREAKDOWN HAS A BDTOORG
''
''
PRIORITY ORDER IS NEW.CALL, TO.DEST.BREAKDOWN,
                  TO.ORG.BREAKDOWN, SLOT.FOR.YEN,
                  NSLOT.FOR.YEN,
                  REQUEST.FOR.SVC, RESPONSE.TO.REQUEST,
                  YEN.ROUTING AND HALT.SIMULATION
''
''
DEFINE INFO AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE SPECINFO AS A 3-DIMENSIONAL INTEGER ARRAY
DEFINE SLOTS.PER.FRAME AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE ATTENUATION AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE LINKCONNECT AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE MEANY.GIVEN AS A 1-DIMENSIONAL REAL ARRAY
DEFINE PROBY.GIVEN AS A 1-DIMENSIONAL REAL ARRAY
DEFINE SLOT.DEPTH AND N AS INTEGER VARIABLES
DEFINE YEN AND DISTANCE AS 2-DIMENSIONAL REAL ARRAYS
DEFINE CLOCK AS A 2-DIMENSIONAL REAL ARRAY
DEFINE LINK.ATTENU AS A 2-DIMENSIONAL REAL ARRAY
DEFINE UP.DATE.PERIOD AS A REAL VARIABLE
DEFINE YENSLOT AS AN INTEGER VARIABLE
DEFINE HOP.GREATEST, HOP.SUM AND HOP.AVG
       AS REAL VARIABLES
DEFINE C.LEVEL AS A 2-DIMENSIONAL REAL ARRAY
DEFINE AVAILCHANNEL AS A 2-DIMENSIONAL REAL ARRAY
DEFINE TOT.HOP.GREATEST AS A REAL VARIABLE
DEFINE N.LINKS AND MAX.LINKS.PER.NODE
       AS INTEGER VARIABLES
DEFINE CALLED.NODE AND CALLING.NODE AS INTEGER VARIABLES
```

```
DEFINE LINKS AND LINK.NODE.RATIO AS REAL VARIABLES
DEFINE NODETRAFFIC AS A REAL VARIABLE
DEFINE BEST.PATH AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE MEANY AS A 2-DIMENSIONAL REAL ARRAY
DEFINE PRNT.COUNTER AS AN INTEGER VARIABLE
DEFINE PATHPRNT AS AN INTEGER VARIABLE
DEFINE CKT.ESTAB, CKT.FAILED, CKT.SUM AND CKT.DISESTAB
       AS INTEGER VARIABLES
DEFINE BK.TO.DEST AND BK.TO.ORG AS INTEGER VARIABLES
DEFINE PRNT AS AN INTEGER VARIABLE
DEFINE TEST.DURATION, SLOT.DURATION, MEAN.SYS.CALL.ARRIV,
      AND MEAN.CALL.DURATION AS REAL VARIABLES
DEFINE FAIR.POINTER AS AN INTEGER VARIABLE
DEFINE LONG.TIME.EST, AVG.P.BD, LONG.P.BD, AVG.C.BD,
       LONG.C.BD AND AVG.TIME.EST AS REAL VARIABLES
DEFINE CKT.LONG.TIME.EST AS AN INTEGER VARIABLE
DEFINE MAX.CKT AS AN INTEGER VARIABLE
DEFINE SUM.BD.TIME, AVG.BD.TIME, TOT.P.BD AND TOT.C.BD
       AS REAL VARIABLES
DEFINE CKTS.BD AS AN INTEGER VARIABLE
DEFINE FRACT.LOST.CALL AND FRACT.SUCCESSFUL.CALL
       AS  REAL VARIABLES
DEFINE C.BD.COUNTER AND P.BD.COUNTER AS INTEGER VARIABLES
DEFINE UPDATELIST AS A 2-DIMENSIONAL INTEGER ARRAY
DEFINE ORG.NODE AND DEST.NODE AS INTEGER VARIABLES
DEFINE SUM.DURATION AND CALL.DURATION AS REAL VARIABLES
DEFINE BREAKTIME AS A REAL VARIABLE
DEFINE NLINK.FOR.NODE AS A 1-DIMENSIONAL INTEGER ARRAY
DEFINE DELAY.SUM AND AVG.DURATION AS REAL VARIABLES
DEFINE OFFERED.TRAFFIC AS A REAL VARIABLE
DEFINE MAX.ATTEMPT AS AN INTEGER VARIABLE
DEFINE ALPHA AS AN INTEGER VARIABLE
DEFINE BEGIN.YEN AS A REAL VARIABLE
DEFINE PARTIAL.BREAKDOWN TO MEAN 3
DEFINE FULL.BREAKDOWN TO MEAN 4
DEFINE DYNAMIC.ALGORITHM AS AN INTEGER VARIABLE
DEFINE CLEAN AS 1-DIMENSIONAL INTEGER ARRAY
DEFINE TOT.YEN.CALLED AS AN INTEGER VARIABLE
DEFINE NSLOT.AVAIL.I AS AN INTEGER VARIABLE
''
END ''PREAMBLE
''
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
MAIN
''
LET LINE.V = 80
''
PRINT 3 LINES AS FOLLOWS
THE OBJECTIVE OF THIS SIMULATION IS TO EVALUATE THE
BEHAVIOR OF THE PROPOSED PACKET RADIO NETWORK USING
YEN ROUTING
SKIP 2 OUTPUT LINES
''
'' THE MAIN PROGRAM CALLS THE FRESH.INPUT ROUTINE THAT
'' SETS THE PARAMETERS FOR SIMULATION
''
LET PRNT.COUNTER = 0
LET FAIR.POINTER = 1
LET TIME.V = 0.000000000
''
PERFORM FRESH.INPUT
''
RESERVE INFO(*,*,*) AS N BY 12 BY 4
RESERVE SPECINFO(*,*,*) AS N BY N BY 12
```

```
RELEASE SEED.V(*)
RESERVE SEED.V(*) AS 10
''
LET SEED.V(1) = 2116429302
LET SEED.V(2) =  683743814
LET SEED.V(3) =  964393174
LET SEED.V(4) = 1217426631
LET SEED.V(5) =  618433579
LET SEED.V(6) = 1157240309
LET SEED.V(7) =   15726055
LET SEED.V(8) =   48108509
LET SEED.V(9) = 1797920909
LET SEED.V(10) = 477424540
''
'' INFO(NODE,SLOT,INDEX) = INTEGER VALUE
''
''    NODE DENOTES NODE NUMBER
''    SLOT DENOTES SLOT NUMBER
''    FOR POSITIVE INTEGER VALUE
''    INDEX = 1  :  TRANSMIT" SLOT WITH CIRCUIT NUMBER
''    INDEX = 2  :  RECEIVE" SLOT FOR RETURN SIGNAL
''    INDEX = 3  :  CALLED OR CALLING NODE NUMBER
''    INDEX = 4  :  TOTAL NUMBER OF RECEIVE SIGNALS
''
''    A PARITICULAR SLOT IS EMPTY IF ITS INTEGER VALUE
''    IS ZERO (NON-TRANSMIT SLOT AND NON-RECEIVE SLOT)
''
''
LET BEGIN.YEN = 200.00
IF DYNAMIC.ALGORITHM EQ 1
   PERFORM DISTANCE.INITIALIZATION
   SCHEDULE A SLOT.FOR.YEN AT BEGIN.YEN
   SCHEDULE A YEN.ROUTING AT BEGIN.YEN + SLOT.DURATION
ALWAYS
''
SCHEDULE A NEW.CALL
AT EXPONENTIAL.F(MEAN.SYS.CALL.ARRIV,5)
SCHEDULE A HALT.SIMULATION AT TEST.DURATION
''
''
START SIMULATION
''
SKIP 2 OUTPUT LINES
PRINT 1 LINE AS FOLLOWS
THE PROGRAM HAS COME TO THE END OF THE SIMULATION
''
STOP
END
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR FRESH.INPUT
''
'' THIS ROUTINE INITIALIZES ALL PARAMETERS FOR
'' THE SIMULATION
''
LET PRNT = 2
LET PATHPRNT = 0
''
'' PRNT HELPS IN DEBUGGING THE SOFTWARE AND PROGRAM LOGIC
''
''  0  == ANNOUNCES EACH PROCESS
''  1  == SELECTIVE PRINTING
''
'' INPUT DATA
''
```

```
LET DYNAMIC.ALGORITHM = 1
LET OFFERED.TRAFFIC = 2.00
LET UP.DATE.PERIOD = 15.0
LET SLOT.DEPTH = 2
LET MEAN.CALL.DURATION = 20.00
LET N = 11
LET TEST.DURATION= 800.00
LET SLOT.DURATION= 0.000010417
LET MAX.CKT = 1500
LET MAX.ATTEMPT = 8 - OFFERED.TRAFFIC
IF MAX.ATTEMPT LT 4
  LET MAX.ATTEMPT = 4
ALWAYS
''
''
'' INITIALIZATION
''
LET CKT.DISESTAB = 0
LET CKT.SUM = 0
LET CKT.ESTAB = 0
LET FRACT.SUCCESSFUL.CALL = 0.0
LET FRACT.LOST.CALL = 0.0
LET BK.TO.DEST = 0
LET BK.TO.ORG = 0
LET BREAKTIME = 12.0 * SLOT.DURATION
LET SUM.DURATION = 0.0
LET AVG.DURATION = 0.0
LET LONG.TIME.EST = 0.0
LET AVG.TIME.EST = 0.0
LET AVG.P.BD = 0.0
LET AVG.C.BD = 0.0
LET LONG.C.BD = 0.0
LET LONG.P.BD = 0.0
LET CKT.LONG.TIME.EST = 0.0
LET AVG.BD.TIME = 0.0
LET SUM.BD.TIME = 0.0
LET CKTS.BD = 0
LET P.BD.COUNTER = 0
LET C.BD.COUNTER = 0
LET TOT.P.BD = 0.0
LET TOT.C.BD = 0.0
LET TOT.YEN.CALLED = 0
LET CKT.FAILED = 0
LET AVG.DURATION = 0.0
LET DELAY.SUM = 0.0
LET ALPHA = 0
''
''
RESERVE LINKCONNECT(*,*) AS N BY N
RESERVE BEST.PATH(*,*) AS N BY N
RESERVE NLINK.FOR.NODE(*) AS N
''
''
FOR I = 1 TO N, DO
  FOR J = 1 TO N, DO
    READ LINKCONNECT(I,J)
  LOOP
LOOP
''
''
FOR I = 1 TO N, DO
  FOR J = 1 TO N, DO
    READ BEST.PATH(I,J)
  LOOP
LOOP
''
PRINT 4 LINES AS FOLLOWS
 THE CONTENTS OF THE INITIAL BEST.PATH MATRIX ARE:
 + TO  =  1   2   3   4   5   6   7   8   9   10   11  =
```

```
 FM +   =                                                      =
 ===============================================================
 FOR I = 1 TO N, DO
 PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2),
       BEST.PATH(I,3), BEST.PATH(I,4), BEST.PATH(I,5),
       BEST.PATH(I,6), BEST.PATH(I,7), BEST.PATH(I,8),
       BEST.PATH(I,9), BEST.PATH(I,10) AND
       BEST.PATH(I,11) AS FOLLOWS
 **   = **  **  **  **  **  **  **  **   **   **   **
 LOOP
 SKIP 2 OUTPUT LINES
''
LET LINKS = 0.0
FOR I = 1 TO N, DO
  FOR J = 1 TO N, DO
    IF LINKCONNECT(I,J) EQ 1
      LET LINKS = LINKS + 1.00
    ALWAYS
  LOOP
LOOP
''
 LET N.LINKS = LINKS / 2.0
 LET LINK.NODE.RATIO = REAL.F(N.LINKS) / REAL.F(N)
''
 PRINT 2 LINE WITH LINKS AND LINK.NODE.RATIO AS FOLLOWS
 TOTAL NUMBER OF LINKS IN THE NETWORK IS ***
 NUMBER OF LINKS PER NODE IS **.**
 SKIP 2 OUTPUT LINE
''
LET NODETRAFFIC = OFFERED.TRAFFIC * LINK.NODE.RATIO
LET MEAN.SYS.CALL.ARRIV = MEAN.CALL.DURATION/NODETRAFFIC
''
LET MAX.LINKS.PER.NODE = 0
''
FOR I = 1 TO N, DO
 LET COUNT = 0
 FOR J = 1 TO N, DO
  IF LINKCONNECT(I,J) EQ 1
  LET COUNT = COUNT + 1
  ALWAYS
 LOOP
''
 IF COUNT GT MAX.LINKS.PER.NODE
  LET MAX.LINKS.PER.NODE = COUNT
 ALWAYS
 LET NLINK.FOR.NODE(I) = COUNT
LOOP
''
RESERVE ATTENUATION(*,*) AS N BY N
RESERVE LINK.ATTENU(*,*) AS N BY N
''
FOR I = 1 TO N, DO
 FOR J = 1 TO N, DO
  IF LINKCONNECT(I,J) EQ 1
   LET PPP      = RANDI.F(1,80,7)
   LET ATTENUATION(I,J) = PPP + 60.0
   LET LINK.ATTENU(I,J) = REAL.F(PPP)
  ALWAYS
  IF LINKCONNECT(I,J) EQ 0
   LET LINK.ATTENU(I,J) = 140.00
  ALWAYS
 LOOP
LOOP
''
''
PRINT 4 LINES WITH TEST.DURATION, SLOT.DURATION,
      MEAN.SYS.CALL.ARRIV AND  MEAN.CALL.DURATION
              AS FOLLOWS
SIMULATION WILL RUN FOR ****.** SECS
```

```
DURATION OF A TIME SLOT IS *.********* SECS
MEAN GENERATION TIME FOR NEW CALL IS ****.** SECS
MEAN DURATION TIME FOR VIRTUAL CIRCUIT IS ****.** SECS
SKIP 1 OUTPUT LINES
''
''
RETURN
END '' FRESH.INPUT
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR DISTANCE.INITIALIZATION
''
'' THE YEN ARRAY HOLDS A REAL POSITIVE NUMBER INDICATING
'' THE  LINK DISTANCE FOR PAIR OF NODES.
'' INITIALLY, IF A DIRECT LINK EXISTS BETWEEN TWO NODES
'' WE SHALL ASSIGN A VALUE OF 1.0  AND IF A DIRECT LINK
'' DOES NOT EXIST, WE SHALL ASSIGN A VALUE OF 9999.0
'' THE VALUES IN THIS ARRAY WILL CHANGE DURING THE
'' SIMULATION AS INIDIVIDUAL LINK WEIGHTS CHANGE TO
'' REFLECT CURRENT STATUS OF LINK AND NODE.
''
''
'' THE DISTANCE ARRAY HOLDS A REAL NON-NEGATIVE NUMBER
'' REPRESENTING THE DISTANCE FROM A NODE TO ANOTHER NODE
''
RESERVE YEN(*,*) AS N BY N
RESERVE CLOCK(*,*) AS N BY N
RESERVE DISTANCE(*,*) AS N BY N
''
FOR I = 1 TO N, DO
 FOR J = 1 TO N, DO
  IF I EQ J
   LET CLOCK(I,J) = 9999.0
   LET DISTANCE(I,J) = 9999.0
   LET YEN(I,J) = 9999.0
  ALWAYS
''
  IF I NE J
   LET CLOCK(I,J) = 800.00
   LET DISTANCE(I,J) = 800.00
   LET YEN(I,J) = 800.00
  ALWAYS
''
  IF LINKCONNECT(I,J) EQ 1
   LET CLOCK(I,J) = 100.00
   LET DISTANCE(I,J) = 100.00
   LET YEN(I,J) = 100.00
  ALWAYS
 LOOP
LOOP
''
'' PRINT ONE OF THESE ARRAYS TO ENSURE
'' THE DISTANCE MATRIX WAS SET UP PROPERLY
''
''
 PRINT 4 LINES AS FOLLOWS
 THE CONTENTS OF THE INITIAL DISTANCE MATRIX ARE:
 + TO =    1      2      3      4      5      6      7  =
 FM+  =                                                 =
 =======================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2),
        DISTANCE(I,3), DISTANCE(I,4),DISTANCE(I,5),
        DISTANCE(I,6) AND DISTANCE(I,7)
              AS FOLLOWS
```

```
 **   + ****.* ****.* ****.* ****.* ****.* ****.* ****.*
 LOOP
 SKIP 1 OUTPUT LINE
 PRINT 4 LINES AS FOLLOWS
 CONTENTS OF THE INITIAL DISTANCE MATRIX (CONT.) :
  + TO  =      8          9          10         11  =
 FM  +  =                                             =
 ==================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9),
        DISTANCE(I,10) AND DISTANCE(I,11) AS FOLLOWS
  **    = ******.*  ******.*  ******.*  ******.*
 LOOP
 SKIP 2 OUTPUT LINES
''
RETURN
END '' DISTANCE INITIALIZATION
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT SLOT.FOR.YEN
''
'' THIS EVENT ALLOCATES SLOTS FOR ROUTING UPDATES
''
DEFINE EFFECTTIME AS A REAL VARIABLE
LET YENSLOT = 1
''
'' YENSLOT = 1 MEANS WE HAVE ALLOCATED SLOTS FOR ROUTING
''              MESSAGES
''
FOR NODEX = 1 TO 11, DO
 FOR NODER = 1 TO 11, DO
  FOR ISLOT = 1 TO 12, DO
    IF INFO(NODEX,ISLOT,1) EQ 0 AND
       INFO(NODER,ISLOT,1) EQ 0 AND
       INFO(NODER,ISLOT,4) LE SLOT.DEPTH
     LET INFO(NODEX,ISLOT,1) = -1
     GO TO OLOOP
    ALWAYS
''
 LOOP
''
'OLOOP'
 LOOP
LOOP
''
 LET EFFECTTIME = 24.0 * SLOT.DURATION
SCHEDULE A NSLOT.FOR.YEN AT TIME.V + EFFECTTIME
SCHEDULE A SLOT.FOR.YEN AT TIME.V + UP.DATE.PERIOD
''
RETURN
END '' SLOT.FOR.YEN
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT NSLOT.FOR.YEN
''
'' THIS EVENT REMOVES SLOTS FOR ROUTING UPDATES
''
IF YENSLOT EQ 1
 LET YENSLOT = 0
 FOR NODEX = 1 TO 11, DO
''
```

```
    FOR ISLOT = 1 TO 12, DO
     IF INFO(NODEX,ISLOT,1) EQ -1
      LET INFO(NODEX,ISLOT,1) = 0
     ALWAYS
    LOOP
''
 LOOP
ALWAYS
''
''
RETURN
END '' NSLOT.FOR.YEN
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT YEN.ROUTING SAVING THE EVENT NOTICE
''
DEFINE MINYEN AS A REAL VARIABLE
''
LET TOT.YEN.CALLED = TOT.YEN.CALLED + 1
''
PRINT 1 LINE WITH TOT.YEN.CALLED AND TIME.V AS FOLLOWS
**** TH ROUTING MANIPULATION INVOKED AT ****.** SECONDS
SKIP 1 OUTPUT LINE
''
'' GET THE CURRENT LINK DISTANCES OF THE NETWORK
''
PERFORM COMPUTE.CURRENT.DISTANCES
''
''
'' THE UPDATELIST ARRAY IS A 2-DIMENSIONAL INTEGER ARRAY
'' USED TO RECORD WHICH NEIGHBOR A NODE JUST RECEIVES "K"
'' MESSAGE FROM
''
RESERVE UPDATELIST(*,*) AS N BY N
''
'' USE THE CURRENT LINK WEIGHT INFORMATION IN THE
'' IMPLEMENTATION OF THE YEN ALGORITHM THAT FOLLOWS.
'' START BY INITIALIZING THE YEN ARRAY.
'' IF THERE IS NO LINK WHICH DIRECTLY CONNECTS TWO NODES,
'' THEN THE LINK WEIGHT IS SET EQUAL TO 9999.0
''
'' WE ALSO READ A COPY OF THE UPDATELIST ARRAY WHICH WILL
'' TBE USED DURING THE YEN.ROUTING EVENT
''
''
'' SEND A "K" MESSAGE FROM A DESTINATION NODE K AND BEGIN
'' TO PERFORM YEN ROUTING
''
FOR K = 1 TO N, DO
 FOR I = 1 TO N, DO
  FOR II = 1 TO N, DO
   FOR JJ = 1 TO N, DO
    LET UPDATELIST(II,JJ) = LINKCONNECT(II,JJ)
   LOOP
  LOOP
  IF UPDATELIST(K,I) EQ 1
   LET UPDATELIST(I,K) = 0
   LET CLOCK(I,K) = DISTANCE(I,K)
   IF CLOCK(I,K) LT YEN(I,K)
    LET BEST.PATH(I,K) = K
    LET YEN(I,K) = DISTANCE(I,K)
   ALWAYS
   FOR J1 = 1 TO N, DO
    IF UPDATELIST(I,J1) EQ 1 AND J1 NE K
     LET UPDATELIST(J1,I) = 0
     LET CLOCK(J1,K) = YEN(I,K) + DISTANCE(J1,I)
     IF CLOCK(J1,K) LT YEN(J1,K)
```

```
   LET YEN(J1,K) = CLOCK(J1,K)
   LET BEST.PATH(J1,K) = I
ALWAYS
FOR J2 = 1 TO N, DO
 IF UPDATELIST(J1,J2) EQ 1 AND (J2 NE I)
    AND (J2 NE K)
    LET UPDATELIST(J2,J1) = 0
    LET CLOCK(J2,K) = YEN(J1,K) + DISTANCE(J2,J1)
    IF CLOCK(J2,K) LT YEN(J2,K)
       LET YEN(J2,K) = CLOCK(J2,K)
       LET BEST.PATH(J2,K) = J1
    ALWAYS
    FOR J3 = 1 TO N, DO
     IF UPDATELIST(J2,J3) EQ 1 AND (J3 NE K) AND
                          (J3 NE I) AND (J3 NE J)
         LET UPDATELIST(J3,J2) = 0
         LET CLOCK(J3,K) = YEN(J2,K)+DISTANCE(J3,J2)
         IF CLOCK(J3,K) LT YEN(J3,K)
          LET YEN(J3,K) = CLOCK(J3,K)
          LET BEST.PATH(J3,K) = J2
         ALWAYS
      FOR J4 = 1 TO N, DO
       IF UPDATELIST(J3,J4) EQ 1 AND (J4 NE J1)
          AND (J4 NE K) AND (J4 NE I) AND (J4 NE J2)
          LET UPDATELIST(J4,J3) = 0
          LET CLOCK(J4,K) = YEN(J3,K)+DISTANCE(J4,J3)
          IF CLOCK(J4,K) LT YEN(J4,K)
           LET YEN(J4,K) = CLOCK(J4,K)
           LET BEST.PATH(J4,K) = J3
          ALWAYS
          FOR J5 = 1 TO N, DO
           IF UPDATELIST(J4,J5) EQ 1
             AND (J5 NE K) AND (J5 NE J1) AND
             (J5 NE J2) AND (J5 NE J3) AND (J5 NE I)
             LET UPDATELIST(J5,J4) = 0
             LET CLOCK(J5,K) =
                       YEN(J4,K) + DISTANCE(J5,J4)
             IF CLOCK(J5,K) LT YEN(J5,K)
               LET YEN(J5,K) = CLOCK(J5,K)
               LET BEST.PATH(J5,K) = J4
             ALWAYS
          FOR J6 = 1 TO N, DO
           IF UPDATELIST(J5,J6) EQ 1
              AND (J6 NE K) AND (J6 NE I) AND
              (J6 NE J1) AND (J6 NE J3) AND
              (J6 NE J3) AND (J6 NE J4) AND
              (J6 NE J2)
             LET UPDATELIST(J6,J5) = 0
             LET CLOCK(J6,K) =
                       YEN(J5,K) + DISTANCE(J6,J5)
             IF CLOCK(J6,K) LT YEN(J6,K)
               LET YEN(J6,K) = CLOCK(J6,K)
               LET BEST.PATH(J6,K) = J5
             ALWAYS
          FOR J7 = 1 TO N, DO
           IF UPDATELIST(J6,J7) EQ 1 AND (J7 NE J5)
              AND (J7 NE K) AND (J7 NE I) AND
              (J7 NE J1) AND (J7 NE J2)
              AND (J7 NE J3) AND (J7 NE J4)
             LET UPDATELIST(J7,J6) = 0
             LET CLOCK(J7,K) =
                       YEN(J6,K) + DISTANCE(J7,J6)
             IF CLOCK(J7,K) LT YEN(J7,K)
               LET YEN(J7,K) = CLOCK(J7,K)
               LET BEST.PATH(J7,K) = J6
             ALWAYS
          FOR J8 = 1 TO N, DO
           IF UPDATELIST(J7,J8) EQ 1 AND (J8 NE J5)
              AND (J8 NE K) AND (J8 NE I) AND
```

```
            (J8 NE J1)
        AND (J8 NE J2) AND (J8 NE J3)
        AND (J8 NE J4) AND (J8 NE J6)
       LET UPDATELIST(J8,J7) = 0
       LET CLOCK(J8,K) =
                 YEN(J7,K) + DISTANCE(J8,J7)
       IF CLOCK(J8,K) LT YEN(J8,K)
         LET YEN(J8,K) = CLOCK(J8,K)
         LET BEST.PATH(J8,K) = J7
     ALWAYS
    FOR J9 = 1 TO N, DO
     IF UPDATELIST(J8,J9) EQ 1 AND (J9 NE J5)
        AND (J9 NE K) AND (J9 NE I)
        AND (J9 NE J1) AND (J9 NE J2)
        AND (J9 NE J3) AND (J9 NE J4)
        AND (J9 NE J6) AND (J9 NE J7)
       LET CLOCK(J9,K) =
                 YEN(J8,K) + DISTANCE(J9,J8)
       IF CLOCK(J9,K) LT YEN(J9,K)
         LET YEN(J9,K) = CLOCK(J9,K)
         LET BEST.PATH(J9,K) = J8
       ALWAYS
      ALWAYS
    LOOP
    LET UPDATELIST(J8,J7) = 1
    ALWAYS
   LOOP
    LET UPDATELIST(J7,J6) = 1
    ALWAYS
   LOOP
    LET UPDATELIST(J6,J5) = 1
    ALWAYS
   LOOP
    LET UPDATELIST(J5,J4) = 1
   ALWAYS
   LOOP
   LET UPDATELIST(J4,J3) = 1
  ALWAYS
 LOOP
  LET UPDATELIST(J3,J2) = 1
 ALWAYS
LOOP
 LET UPDATELIST(J2,J1) = 1
ALWAYS
LOOP
   LET UPDATELIST(J1,I) = 1
 ALWAYS
 LOOP
ALWAYS
LOOP
LOOP
''
IF PATHPRNT EQ 1
 PRINT 4 LINES AS FOLLOWS
 THE CONTENTS OF THE  MANIPULATED YEN MATRIX ARE:
 + TO =   1      2      3      4      5      6      7  =
 FM + =                                                =
 ======================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, YEN(I,1), YEN(I,2), YEN(I,3),
        YEN(I,4), YEN(I,5),YEN(I,6) AND YEN(I,7) AS
        FOLLOWS
 **   + ****.* ****.* ****.* ****.* ****.* ****.* ****.*
 LOOP
 SKIP 1 OUTPUT LINE
 PRINT 5 LINES AS FOLLOWS
 CONTENTS OF THE MANIPULATED YEN MATRIX (CONT.) :
 +     =                                           =
   + TO =      8         9         10        11    =
```

```
 FM  +  =                                               =
 ================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, YEN(I,8), YEN(I,9), YEN(I,10),
        YEN(I,11) AS FOLLOWS
  **   =  ******.* ******.*  ******.*  ******.*
 LOOP
 SKIP 2 OUTPUT LINES
''
''
 PRINT 5 LINES AS FOLLOWS
 THE CONTENTS OF THE MANIPULATED BEST.PATH MATRIX ARE:
 + TO   =  1   2   3   4   5   6   7   8   9   10   11 =
   +    =                                               =
 FM   + =                                               =
 =======================================================
 FOR I = 1 TO N, DO
 PRINT 1 LINE WITH I, BEST.PATH(I,1), BEST.PATH(I,2),
              BEST.PATH(I,3),BEST.PATH(I,4),
              BEST.PATH(I,5), BEST.PATH(I,6),
              BEST.PATH(I,7), BEST.PATH(I,8),
              BEST.PATH(I,9), BEST.PATH(I,10) AND
              BEST.PATH(I,11) AS FOLLOWS
  **   = **  **  **  **  **  **  **  **   **   **  **
 LOOP
 SKIP 2 OUTPUT LINES
ALWAYS
''
SCHEDULE A YEN.ROUTING AT TIME.V + UP.DATE.PERIOD
''
RETURN
END '' YEN.ROUTING
''
''
''
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
ROUTINE FOR PATH.UPDATE GIVEN K, PREVIOUS AND KTRAVEL
''
  FOR J = 1 TO N, DO
    IF UPDATELIST(PREVIOUS,J) EQ 1
       LET UPDATELIST(J,PREVIOUS) = 0
       LET CLOCK(J,K) = YEN(PREVIOUS,K)
                      + DISTANCE(J,PREVIOUS)
       IF CLOCK(J,K) LE YEN(J,K)
           LET YEN(J,K) = CLOCK(J,K)
           LET BEST.PATH(J,K) = I
       ALWAYS
    ALWAYS
''LOOP
''
RETURN
''
END '' PATH.UPDATE
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
ROUTINE TO COMPUTE.CURRENT.DISTANCES
''
DEFINE ARG1,ARG2 AND ARG3 AS REAL VARIABLES
DEFINE MEANY.GIVEN.NODES AS A REAL VARIABLE
DEFINE SUMMATION AS A REAL VARIABLE
''
RESERVE C.LEVEL(*,*) AS N BY N
```

```
RESERVE AVAILCHANNEL(*,*) AS N BY N
RESERVE MEANY(*,*) AS N BY N
RESERVE MEANY.GIVEN(*)   AS 12
RESERVE PROBY.GIVEN(*)   AS 12
''
FOR I = 1 TO N, DO
 FOR J = 1 TO N, DO
''
  IF I EQ J
   LET C.LEVEL(I,J) = 9999.0
   LET DISTANCE(I,J) = 9999.0
''ALWAYS
''
  IF I NE J
   LET C.LEVEL(I,J) = 800.00
   LET DISTANCE(I,J) = 800.00
''ALWAYS
''
''IF LINKCONNECT(I,J) EQ 1
''
    LET NSLOT.AVAIL.I = 0
    FOR K = 1 TO 12, DO
     IF INFO(J,K,1) LE 0 AND INFO(I,K,1) LE 0
        AND INFO(I,K,4) EQ 0
      LET NSLOT.AVAIL.I = NSLOT.AVAIL.I + 1
     ALWAYS
    LOOP
''
    LET MTJ = 0
    FOR KK = 1 TO 12, DO
     IF INFO(J,KK,1) GT 0
      LET MTJ = MTJ + 1
     ALWAYS
    LOOP
''
    IF NSLOT.AVAIL.I EQ 0
      LET C.LEVEL(I,J) = 400.0
      LET DISTANCE(I,J)    = 400.0
      GO TO OTHER.IJ
    ALWAYS
''
    IF MTJ EQ 0
      LET AVAILCHANNEL(I,J) = 2*REAL.F(NSLOT.AVAIL.I)/3.0
      LET C.LEVEL(I,J) = 81.0 / AVAILCHANNEL(I,J) - 10.1
      LET DISTANCE(I,J) =
          C.LEVEL(I,J) + ALPHA * LINK.ATTENU(I,J)
      GO TO OTHER.IJ
    ALWAYS
''
''
''
    LET NJ = 12 - MTJ
    LET MEANY.GIVEN.NODES = 0.0
''
    LET LARGEST.MRJ  = MTJ
    IF MTJ GE 6
      LET LARGEST.MRJ = NJ
    ALWAYS
''
    LET SMALLEST.MRJ = TRUNC.F((MTJ + 1.0) / 2.0)
    IF SMALLEST.MRJ LT 1
      LET SMALLEST.MRJ = 1
    ALWAYS
''
    FOR MRJ = SMALLEST.MRJ TO LARGEST.MRJ, DO
''
      LET MEANY.GIVEN(MRJ) = 0.0
''
      LET MAXY = MRJ
```

```
        IF MRJ GT NSLOT.AVAIL.I
           LET MAXY = NSLOT.AVAIL.I
        ALWAYS
''
''
        LET SUMMATION = 0.0
        LET AIY = 1
        LET BIY = MAXY + 1
''
        FOR IIY = AIY TO BIY, DO
''
          LET IY = IIY - 1
''
          LET TOP1 = NJ - NSLOT.AVAIL.I
          LET BOTTOM1 = MRJ - IY
          IF BOTTOM1 GT TOP1
            LET ARG1 = 0.0
          ALWAYS
''
          LET TOP2 = NSLOT.AVAIL.I
          LET BOTTOM2 = IY
          IF BOTTOM2 GT TOP2
            LET ARG2 = 0.0
          ALWAYS
''
          IF BOTTOM2 GT TOP2 OR BOTTOM1 GT TOP1
             GO TO OTHERIY
          ALWAYS
''
          PERFORM COMBINATION GIVEN TOP1 AND BOTTOM1
                  YIELDING ARG1
          PERFORM COMBINATION GIVEN TOP2 AND BOTTOM2
                  YIELDING ARG2
''
'OTHERIY'
''
          LET PROBY.GIVEN(IIY) = ARG1 * ARG2
          LET SUMMATION    = SUMMATION + PROBY.GIVEN(IIY)
''
          LET MEANY.GIVEN(MRJ) =
           REAL.F(IY) * PROBY.GIVEN(IIY) + MEANY.GIVEN(MRJ)
''
        LOOP
''
        IF SUMMATION LT 0.001
          LET MEANY.GIVEN(MRJ) = 0.000
          GO TO OTHERMRJ
        ALWAYS
''
        LET MEANY.GIVEN(MRJ) = MEANY.GIVEN(MRJ)/SUMMATION
''
'OTHERMRJ'
''
        LET MEANY.GIVEN.NODES = MEANY.GIVEN(MRJ)
                                + MEANY.GIVEN.NODES
     LOOP
''
     LET R   = LARGEST.MRJ - SMALLEST.MRJ + 1
''
     LET MEANY(I,J) = MEANY.GIVEN.NODES / REAL.F(R)
     LET AVAILCHANNEL(I,J) =
         2.0 *(REAL.F(NSLOT.AVAIL.I) - MEANY(I,J))/3.0
''
''
     IF AVAILCHANNEL(I,J) LT 0.20
      LET C.LEVEL(I,J) = 395.0
      LET DISTANCE(I,J)  = 395.0
      GO TO OTHER.IJ
     ALWAYS
```

```
''
''
    LET C.LEVEL(I,J) = 81.0 / AVAILCHANNEL(I,J) - 10.1
    LET DISTANCE(I,J) = C.LEVEL(I,J)
                    + ALPHA * LINK.ATTENU(I,J)
''
  ALWAYS
'OTHER.IJ'
 LOOP
''
LOOP
''
''
IF PATHPRNT EQ 1
 PRINT 5 LINES AS FOLLOWS
 THE CONTENTS OF THE DISTANCE ARRAY AFTER UPDATING
 THE LINK DISTANCES ARE:
 + TO =    1       2       3       4       5       6       7  =
 FM + =                                                       =
 ==============================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,1), DISTANCE(I,2),
        DISTANCE(I,3),DISTANCE(I,4),DISTANCE(I,5),
        DISTANCE(I,6) AND DISTANCE(I,7) AS FOLLOWS
  **   + ****.* ****.* ****.* ****.* ****.* ****.*  ****.*
 LOOP
 SKIP 1 OUTPUT LINE
 PRINT 4 LINES AS FOLLOWS
 CONTENTS OF THE DISTANCE ARRAY(CONT.) :
  + TO =      8         9         10         11    =
 FM +  =                                           =
 ===================================================
 FOR I = 1 TO N, DO
  PRINT 1 LINE WITH I, DISTANCE(I,8), DISTANCE(I,9),
        DISTANCE(I,10) AND DISTANCE(I,11) AS FOLLOWS
  **   = ******.*  ******.*  ******.*  ******.*
 LOOP
 SKIP 2 OUTPUT LINES
ALWAYS
''
RETURN
END '' COMPUTE.CURRENT.DISTANCES
''
''
''
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
ROUTINE FOR COMBINATION GIVEN TOP AND BOTTOM
YIELDING ANS
''
DEFINE ANS1, ANS2 AND ANS3 AS REAL VARIABLES
DEFINE CRESULT AS A REAL VARIABLE
''
LET ITOP = TOP
LET IBOTTOM = BOTTOM
LET IC = ITOP - IBOTTOM
''
PERFORM FACTORIAL GIVEN ITOP YIELDING ANS1
PERFORM FACTORIAL GIVEN IBOTTOM YIELDING ANS2
PERFORM FACTORIAL GIVEN IC YIELDING ANS3
''
LET CRESULT = ANS1 / (ANS3 * ANS2)
LET ANS = CRESULT
''
RETURN
END '' COMBINATION
```

```
''
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
ROUTINE FOR FACTORIAL GIVEN IVALUE YIELDING FAC.VALUE
''
DEFINE FAC.VALUE AS A REAL VARIABLE
DEFINE INUM AS A INTEGER VARIABLE
''
LET INUM = IVALUE
IF INUM EQ 1 OR INUM EQ 0
  LET FAC.VALUE = 1.0
  RETURN
ALWAYS
''
IF INUM GE 2
  LET FAC.VALUE = 1.0
  FOR I = 2 TO INUM, DO
     LET FAC.VALUE = FAC.VALUE * REAL.F(I)
  LOOP
  RETURN
ALWAYS
''
PRINT 1 LINE AS FOLLOWS
YOUR INPUT DATA TO THE FACTORIAL ROUTINE IS NEGATIVE
PERFORM TERMINATION
''
RETURN
END '' FACTORIAL
''
''
''
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
EVENT NEW.CALL SAVING THE EVENT NOTICE
''
''
'' THIS EVENT GENERATES CALL AND SENDS "REQUEST FOR
'' SERVICE" FROM A CALLING NODE TO A CALLED NODE
''
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 NEW CALL GENERATED AT TIME ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
DEFINE DELAY1 AS A REAL VARIABLE
''
LET CKT.SUM = CKT.SUM + 1
''
IF CKT.SUM GE MAX.CKT
 PRINT 2 LINES WITH TIME.V AND MAX.CKT AS FOLLOWS
 NUMBER OF CKTS ATTEMPTED EXCEEDS **** CKTS PERMITTED.
 SIMULATION HALTED AT ****.*** SEC
 SKIP 1 OUTPUT LINE
''
 PERFORM TERMINATION
 RETURN
ALWAYS
''
SCHEDULE A NEW.CALL
AT TIME.V + EXPONENTIAL.F(MEAN.SYS.CALL.ARRIV,5)
''
```

```
'' SELECT A TRANSMITTER

'SELECTAGAIN'

LET XMTR = RANDI.F(1,N,1)

'' SELECT A CORRESPONDING RECEIVER
''
LET RCVR = RANDI.F(1,N,2)

IF RCVR EQ XMTR
 GO TO SELECTAGAIN
ALWAYS

LET ORG.NODE = XMTR
LET DEST.NODE = RCVR
LET CALLED.NODE = BEST.PATH(XMTR,RCVR)

IF CALLED.NODE LT 1 OR CALLED.NODE GT 12
 PRINT 1 LINE AS FOLLOWS
 CALLED.NODE WAS NOT DETERMINED PROPERLY
 PERFORM TERMINATION
ALWAYS
''
 PRINT 1 LINE WITH CKT.SUM,ORG.NODE,DEST.NODE AND TIME.V
                    AS FOLLOWS
 CIRCUIT *** FROM NODE ** TO ** BEGUN AT ****.**** SECS
 SKIP 1 OUTPUT LINE

LET BK.TO.DEST = BK.TO.DEST + 1

FOR J = 1 TO 12 , DO
 IF INFO(XMTR,J,1) EQ 0 AND INFO(XMTR,J,4) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0
    GO TO 'ON1'
 ALWAYS
LOOP

IF PRNT EQ 0
 PRINT 3 LINES WITH ORG.NODE,CALLED.NODE AND CKT.SUM
                    AS FOLLOWS
 NO MUTUALLY AVAILABLE SLOTS BETWEEN THE ORG.NODE ** AND
 CALLED NODE ** TO CARRY THE REQUEST SERVICE MESSAGE FOR
 CIRCUIT NUMBER *****
 SKIP 1 OUTPUT LINE
ALWAYS

LET CKT.FAILED = CKT.FAILED + 1
LET BK.TO.DEST = BK.TO.DEST - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
GO TO LAST.NEW.CALL

'' SELECTS A CURRENT SLOT RANDOMLY AND
'' CONTINUES PROCESSING

'ON1'

LET CURRENT.SLOT = RANDI.F(1,12,3)
''
'' FINDS THE NEXT MUTUALLY AVAILABLE SLOT
''
LET SLOT1 = 0
LET FRAME1 = 0

IF CURRENT.SLOT EQ 12
```

```
 GO TO NEXT.FRAME1
ALWAYS
''
LET K = CURRENT.SLOT + 1
''
FOR J = K TO 12 , DO
IF INFO(ORG.NODE,J,1) GT 0 OR INFO(ORG.NODE,J,4) GT 0
 LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
LET FRAME1 = 1
FOR J = 1 TO CURRENT.SLOT , DO
IF INFO(ORG.NODE,J,1) GT 0 OR INFO(ORG.NODE,J,4) GT 0
 LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
LET FRAME1 = 0
FOR J = K TO 12 , DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
''
LET FRAME1 = 1
FOR J = 1 TO CURRENT.SLOT, DO
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
GO TO YY
''
''
'NEXT.FRAME1'
''
LET FRAME1 = 1
FOR J = 1 TO 12, DO
IF INFO(ORG.NODE,J,1) GT 0 OR INFO(ORG.NODE,J,4) GT 0
 LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
ALWAYS
 IF SPECINFO(ORG.NODE,CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
  LET SLOT1 = J
  LET SPECINFO(ORG.NODE,CALLED.NODE,J) = 0
  GO TO ON2
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
```

```
IF INFO(ORG.NODE,J,1) EQ 0 AND INFO(ORG.NODE,J,4) EQ 0
   AND INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT1 = J
 GO TO ON2
ALWAYS
LOOP
''
'YY'
''
PRINT 1 LINE WITH CKT.SUM AS FOLLOWS
CIRCUIT NO.**** FAILED IN EVENT NEW CALL
SKIP 2 OUTPUT LINES
''
LET CKT.FAILED = CKT.FAILED + 1
LET BK.TO.DEST = BK.TO.DEST - 1
LET P.BD.COUNTER = P.BD.COUNTER + 1
RETURN
''
'' ON2 IDENTIFIES A SLOT TO CARRY THE SERVICE MESSAGE
'' TO THE CALLED NODE AND CREATES THE SERVICE MESSAGE
''
'ON2'
''
CREATE A MESSAGE
''
LET CKT.NUMBER(MESSAGE) = CKT.SUM
LET TYPE(MESSAGE) = 1
LET ORIGINATOR(MESSAGE) = ORG.NODE
LET DESTINATION(MESSAGE) = DEST.NODE
LET FM.NODE(MESSAGE) = ORG.NODE
LET TO.NODE(MESSAGE) = CALLED.NODE
LET START.TIME(MESSAGE) = TIME.V
LET SLOT.ARRIVAL(MESSAGE) = SLOT1
LET SLOT.ASSIGN(MESSAGE) = SLOT1
LET RECSLOT(MESSAGE) = SLOT1
LET DIRECTION(MESSAGE) = 0
LET REATTEMPT(MESSAGE) = 1
''
IF PRNT EQ 0
 PRINT 2 LINES WITH SLOT1 AND FRAME1 AS FOLLOWS
 SLOT ** OF FRAME ** WAS USED TO CARRY REQUEST FOR
 SERVICE FROM CALLING NODE TO THE CALLED NODE
 SKIP 1 OUTPUT LINE
ALWAYS
''
'' CALCULATES WHEN THE SERVICE MESSAGE WILL ARRIVE AT
'' THE CALLED NODE AND SCHEDULES ITS ARRIVAL
''
''
LET DELAY1 = (REAL.F(12*FRAME1 + SLOT1 - CURRENT.SLOT))
                * SLOT.DURATION
''
IF PRNT EQ 0
 PRINT 2 LINES WITH CKT.SUM,CALLED.NODE AND
      (TIME.V + DELAY1) AS FOLLOWS
 CKT ***** HAS SCHEDULED AN REQUEST FOR SVC AT NODE **
 AT TIME ***.****
 SKIP 2 OUTPUT LINES
ALWAYS
''
''  MAKE THOSE SLOTS ALLOCATED FOR ROUTING MESSAGES
''  AVAILABLE FOR VOICE TRAFFIC
''
''
SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
AT TIME.V + DELAY1
''
'LAST.NEW.CALL'
 IF PRNT EQ 0
```

```
  PRINT 1 LINE AS FOLLOWS
  ATTRIBUTES OF ENTITY AT THE END OF NEW CALL ARE :
  LIST ATTRIBUTES OF MESSAGE
  SKIP 2 OUTPUT LINES
 ALWAYS
''
''
RETURN
END '' NEW.CALL
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT REQUEST.FOR.SVC GIVEN MSG1 SAVING THE EVENT NOTICE
''
''  THIS EVENT SIMULATES ACTIONS PERFORMED AT A CALLED
''  NODE AFTER RECEIVING AN REQUEST FOR SERVICE FROM A
''  CALLING NODE
''
LET MESSAGE = MSG1
''
IF  PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
REQUEST.FOR.SVC PERFORMED AT TIME  ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY AT START OF REQUEST.FOR.SVC ARE:
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
''
ALWAYS
''
DEFINE DELAY2 AND DELAYR AS  REAL VARIABLES
''
LET FRAME.REC = 0
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
LET SLOT.REC = CURRENT.SLOT
''
FOR I = 1 TO 12, DO
 IF INFO(CALLED.NODE,I,1) EQ CKT.NUMBER(MESSAGE)
  PRINT 1 LINE AS FOLLOWS
  THE ROUTING IS NOT LOOP FREE
  PERFORM TERMINATION
 ALWAYS
LOOP
''
''
IF INFO(CALLED.NODE,SLOT.REC,4) LT SLOT.DEPTH AND
   INFO(CALLED.NODE,SLOT.REC,1) EQ 0
 GO TO OK1
ALWAYS
''
IF REATTEMPT(MESSAGE) LT MAX.ATTEMPT
 LET REATTEMPT(MESSAGE) = REATTEMPT(MESSAGE) + 1
 LET SLOT.USED = SLOT.REC
''
 LET FRAMER = 1
 LET IIR = SLOT.USED + 1
 FOR IR = IIR TO 12, DO
  IF INFO(CALLING.NODE,IR,1) GT 0 OR
     INFO(CALLING.NODE,IR,4) GT 0
   LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
  ALWAYS
```

```
 IF SPECINFO(CALLING.NODE,CALLED.NODE,IR) EQ 6 AND
    INFO(CALLED.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4) EQ 0
   LET SLOTR = IR
   LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
   GO TO MORE.ATTEMPT
  ALWAYS
LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 LET LJR = SLOT.USED - 1
 FOR JR = 1 TO LJR, DO
 IF INFO(CALLING.NODE,JR,1) GT 0 OR
    INFO(CALLING.NODE,JR,4) GT 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,JR) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,CALLED.NODE,JR) EQ 6 AND
    INFO(CALLED.NODE,JR,1) EQ 0 AND
    INFO(CALLING.NODE,JR,1) EQ 0 AND
    INFO(CALLING.NODE,JR,4) EQ 0
  LET SLOTR = JR
  LET SPECINFO(CALLING.NODE,CALLED.NODE,JR) = 0
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
 LET FRAMER = 1
 FOR IR = (SLOT.USED + 1) TO 12, DO
 IF INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4) EQ 0 AND
    INFO(CALLED.NODE,IR,1) EQ 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR JR = 1 TO (SLOT.USED - 1), DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
     INFO(CALLING.NODE,JR,4) EQ 0 AND
     INFO(CALLED.NODE,JR,1) EQ 0
   LET SLOTR = JR
   GO TO MORE.ATTEMPT
  ALWAYS
 LOOP
 GO TO XX
''
 'MORE.ATTEMPT'
''
 LET DELAYR = (REAL.F(SLOTR - SLOT.USED) + FRAMER*12.0)
               * SLOT.DURATION
 LET RECSLOT(MESSAGE) = SLOTR
 LET DIRECTION(MESSAGE) = 0
 LET SLOT.ASSIGN(MESSAGE) = SLOTR
 LET SLOT.ARRIVAL(MESSAGE) = SLOTR
''
 SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
 AT TIME.V + DELAYR
''
 RETURN
```

```
ALWAYS
''
'XX'
''
IF PRNT EQ 0
PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
      ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
      TIME.V, TO.NODE(MESSAGE) AND FM.NODE(MESSAGE)
      AS FOLLOWS
CKT **** FM ** TO ** BROKEDOWN AT TIME ****.****** DUE TO
NO MUTUALLY AVAILABLE SLOT BETWEEN THE CALLED NODE **
AND THE CALLING NODE **
SKIP 2 OUTPUT LINE
ALWAYS
''
'EXIT'
''
LET CKT.FAILED = CKT.FAILED + 1
LET BK.TO.DEST = BK.TO.DEST - 1
''
IF ORIGINATOR(MESSAGE) EQ FM.NODE(MESSAGE)
 LET P.BD.COUNTER = P.BD.COUNTER + 1
 DESTROY THE MESSAGE CALLED MESSAGE
 RETURN
ALWAYS
''
LET BK.TO.ORG = BK.TO.ORG + 1
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET DIRECTION(MESSAGE) = 3
LET START.TIME(MESSAGE) = TIME.V
''
 SCHEDULE A TO.ORG.BREAKDOWN GIVEN MESSAGE
 AT TIME.V + BREAKTIME
 RETURN
''
''
'' FIND THE NEXT MUTUALLY AVAILABLE SLOT
''
''
'OK1'
''
LET SLOT2 = 0
LET FRAME2 = 0
''
IF CURRENT.SLOT EQ 12
 GO TO NEXT.FRAME2
ALWAYS
''
LET L = CURRENT.SLOT + 1
''
FOR J = L TO 12 , DO
IF INFO(CALLED.NODE,J,1) GT 0 OR
   INFO(CALLED.NODE,J,4) GT 0
 LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
ALWAYS
IF SPECINFO(CALLED.NODE,CALLING.NODE,J) EQ 6 AND
   INFO(CALLING.NODE,J,1) EQ 0 AND
   INFO(CALLED.NODE,J,1) EQ 0 AND
   INFO(CALLED.NODE,J,4) EQ 0
 LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
''
LET FRAME2 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLED.NODE,J,1) GT 0 OR
    INFO(CALLED.NODE,J,4) GT 0
```

```
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLED.NODE,CALLING.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
''
LET FRAME2 = 0
FOR J = L TO 12, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
''
LET FRAME2 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
GO TO YYY
''
'NEXT.FRAME2'
''
LET FRAME2 = 1
FOR J = 1 TO 12, DO
 IF INFO(CALLED.NODE,J,1) GT 0 OR
    INFO(CALLED.NODE,J,4) GT 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLED.NODE,CALLING.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ C AND
    INFO(CALLED.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
  LET SLOT2 = J
  GO TO OK2
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
 LET SLOT2 = J
 GO TO OK2
ALWAYS
LOOP
''
'YYY'
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** FAILED IN EVENT REQUEST.FOR.SVC
SKIP 1 OUTPUT LINE
''
GO TO EXIT
''
'' OK2 IDENTIFIES THE SLOT TO CARRY THE SLOT ASSIGNMENT
```

```
'' AND SENDS REQUEST BACK TO THE CALLING NODE AND ALSO
'' COMPUTES WHEN THE SERVICE MESSAGE WILL ARRIVE AT
'' THE CALLING NODE
''
''
'OK2'
''
LET DELAY2 = (REAL.F(12*FRAME2 + SLOT2 - CURRENT.SLOT))
                * SLOT.DURATION
''
'' ASSIGNS SLOTS,UPDATES MESSAGE AND
'' SCHEDULES RESPONSE.TO.REQUEST AT
'' THE CALLED NODE
''
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT2
LET SLOT.ASSIGN(MESSAGE) = SLOT.REC
LET RECSLOT(MESSAGE) = SLOT.REC
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE), FM.NODE(MESSAGE)
        AND (TIME.V + DELAY2) AS FOLLOWS
CIRCUIT **** HAS SCHEDULED A RESPONSE TO SVC AT NODE **
AT TIME ****.****
SKIP 2 OUTPUT LINES
''
PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT END OF REQUEST FOR SVC ARE:
LIST ATTRIBUTES OF MESSAGE
SKIP 1 OUTPUT LINE
ALWAYS
''
SCHEDULE A RESPONSE.TO.REQUEST GIVEN MESSAGE
AT TIME.V + DELAY2
''
RETURN
END  '' REQUEST FOR SERVICE
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT RESPONSE.TO.REQUEST GIVEN MSG2
''
''  THIS EVENT SIMULATES ACTIONS PERFORMED AT A CALLING
''  NODE AFTER RECEIVING A RESPONSE TO REQUEST FROM
''  A CALLED NODE
''
''
LET MESSAGE = MSG2
''
IF  PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 RESPONSE.TO.REQUEST PERFORMED AT TIME  ****.*****
''
 SKIP 2 OUTPUT LINES
ALWAYS
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY AT START OF RTR ARE:
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
''
ALWAYS
''
DEFINE DELAY3 AND DELAYR AS REAL VARIABLES
''
```

```
LET FRAME.REC = 0
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
LET SLOT.REC = SLOT.ARRIVAL(MESSAGE)
''
IF INFO(CALLING.NODE,SLOT.REC,1) EQ 0
.   AND INFO(CALLING.NODE,SLOT.REC,4) LT SLOT.DEPTH
 GO TO CORRECT
 ALWAYS
''
IF REATTEMPT(MESSAGE) LT MAX.ATTEMPT
 LET REATTEMPT(MESSAGE) = REATTEMPT(MESSAGE) + 1
 LET SLOT.USED = RECSLOT(MESSAGE)
''
 LET FRAMER = 1
 FOR IR = (SLOT.USED + 1) TO 12, DO
 IF INFO(CALLING.NODE,IR,1) GT 0 OR
    INFO(CALLING.NODE,IR,4) GT 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,CALLED.NODE,IR) EQ 6 AND
    INFO(CALLED.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,1) EQ 0 AND
    INFO(CALLING.NODE,IR,4) EQ 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR IR = 1 TO (SLOT.USED - 1), DO
 IF INFO(CALLING.NODE,IR,1) GT 0 OR
    INFO(CALLING.NODE,IR,4) GT 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
 ALWAYS
''
 IF SPECINFO(CALLING.NODE,CALLED.NODE,IR) EQ 6 AND
    INFO(CALLED.NODE,IR,1) EQ 0 AND
   INFO(CALLING.NODE,IR,1) EQ 0 AND
   INFO(CALLING.NODE,IR,4) EQ 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,IR) = 0
  LET SLOTR = IR
  GO TO MORE.ATTEMPT
 ALWAYS
 LOOP
''
''
 LET FRAMER = 1
 FOR JR = (SLOT.USED + 1) TO 12, DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
     INFO(CALLING.NODE,JR,4) EQ 0 AND
     INFO(CALLED.NODE,JR,1) EQ 0
   LET SLOTR = JR
   GO TO MORE.ATTEMPT
  ALWAYS
 LOOP
''
IF SLOT.USED EQ 1
 GO TO XX
ALWAYS
''
 LET FRAMER = 2
 FOR JR = 1 TO (SLOT.USED - 1), DO
  IF INFO(CALLING.NODE,JR,1) EQ 0 AND
```

```
      INFO(CALLING.NODE,JR,4) EQ 0 AND
      INFO(CALLED.NODE,JR,1) EQ 0
    LET SLOTR = JR
    GO TO MORE.ATTEMPT
   ALWAYS
  LOOP
  GO TO XX
''
'MORE.ATTEMPT'
''
  LET DELAYR = (REAL.F(SLOTR - SLOT.REC)+(FRAMER * 12.0))
                  * SLOT.DURATION
  LET RECSLOT(MESSAGE) = 0
  LET SLOT.ASSIGN(MESSAGE) = 0
  LET DIRECTION(MESSAGE) = 0
  LET SLOT.ARRIVAL(MESSAGE) = SLOTR
''
  SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
  AT TIME.V + DELAYR
''
  RETURN
ALWAYS
''
'XX'
''
IF PRNT EQ 0
PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
      ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
      TIME.V, FM.NODE(MESSAGE) AND TO.NODE(MESSAGE)
              AS FOLLOWS
CKT **** FM ** TO ** BROKE DOWN AT ****.****** DUE TO
NO MUTUALLY AVAILABLE SLOT BETWEEN THE CALLED NODE **
AND THE CALLING NODE **
SKIP 2 OUTPUT LINES
ALWAYS
''
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET CKT.FAILED = CKT.FAILED + 1
LET BK.TO.DEST = BK.TO.DEST - 1
LET BK.TO.ORG = BK.TO.ORG + 1
LET START.TIME(MESSAGE) = TIME.V
''
IF PRNT EQ 0
 PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AND TIME.V
       AS FOLLOWS
 CIRCUIT ***** FAILED TO CONNECT AT TIME ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
''
IF FM.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
 LET RECSLOT(MESSAGE) = 13
 LET DIRECTION(MESSAGE) = 0
''
 SCHEDULE A TO.DEST.BREAKDOWN GIVEN MESSAGE
 AT TIME.V + BREAKTIME
 RETURN
ALWAYS
''
''
 LET DIRECTION(MESSAGE) = 4
''
 SCHEDULE A TO.ORG.BREAKDOWN GIVEN MESSAGE
 AT TIME.V + BREAKTIME
''
 RETURN
''
```

```
''
'CORRECT'
''
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),1)
     = CKT.NUMBER(MESSAGE)
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),2) = SLOT.REC
LET INFO(CALLING.NODE,SLOT.ASSIGN(MESSAGE),3)=CALLED.NODE
LET TSLOT = SLOT.ASSIGN(MESSAGE)
LET SPECINFO(CALLING.NODE,CALLED.NODE,TSLOT)
     = 0
LET INFO(CALLING.NODE,SLOT.REC,4)
     = INFO(CALLING.NODE,SLOT.REC,4) + 1
LET XSLOT.CALLED = SLOT.REC
LET RSLOT.CALLED = SLOT.ASSIGN(MESSAGE)
LET SLOT.ARRIVAL(MESSAGE) = RSLOT.CALLED
LET SLOT.ASSIGN(MESSAGE) = XSLOT.CALLED
LET RECSLOT(MESSAGE) = XSLOT.CALLED
''
''
LET INFO(CALLED.NODE,XSLOT.CALLED,1)=CKT.NUMBER(MESSAGE)
LET INFO(CALLED.NODE,XSLOT.CALLED,2)=RSLOT.CALLED
LET INFO(CALLED.NODE,XSLOT.CALLED,3)=CALLING.NODE
LET SPECINFO(CALLED.NODE,CALLING.NODE,XSLOT.CALLED) = 0
LET INFO(CALLED.NODE,RSLOT.CALLED,4)
     = INFO(CALLED.NODE,RSLOT.CALLED,4)+1
''
''  CHECK WHETHER THE CIRCUIT IS COMPLETE
''  IF YES, CALL THE COMPLETE.CKT ROUTINE AND
''  COLLECT STATISTICAL DATA
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM VIRTUAL.CKT GIVEN MESSAGE
 RETURN
ALWAYS
''
''  IF THE CKT HAS NOT BEEN ESTABLISHED ALL THE WAY TO
''  THE DESTINATION ,THEN SPECIAL ACTION MUST BE TAKEN
''  TO ESTABLISH THE NEXT LINK TO THE DESTINATION
''
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
LET TO.NODE(MESSAGE)
     = BEST.PATH(FM.NODE(MESSAGE),DESTINATION(MESSAGE))
''
''
''  THE REST OF THIS EVENT SIMULATES ACTIONS PERFORMED
''  AT AN INTERMEDIATE NODE .
''
''  WE BEGIN TO CHECK WHETHER THERE IS A SLOT AVAILABLE
''  IN THIS ASSIGNED CALLING NODE TO ACCOMODATE
''  THE TRANSMISSION TO THE NEWLY ASSIGNED CALLED NODE
''
LET CALLING.NODE = FM.NODE(MESSAGE)
LET CALLED.NODE = TO.NODE(MESSAGE)
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
LET SLOT3 = 0
LET FRAME3 = 0
''
IF CURRENT.SLOT EQ 12
 LET K = 1
 GO TO NEXT.FRAME3
ALWAYS
''
LET K = CURRENT.SLOT + 1
FOR J = K TO 12 , DO
 IF INFO(CALLED.NODE,J,1) GT 0 OR
     INFO(CALLING.NODE,J,4) GT 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,J) = 0
```

```
 ALWAYS
 IF SPECINFO(CALLED.NODE,CALLING.NODE,J) EQ 6 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
  LET SPECINFO(CALLING.NODE,CALLED.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
LET FRAME3 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLING.NODE,J,1) GT 0 OR
    INFO(CALLING.NODE,J,4) GT 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
LET FRAME3 = 0
FOR J = K TO 12 , DO
 IF INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
''
LET FRAME3 = 1
FOR J = 1 TO CURRENT.SLOT, DO
 IF INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
GO TO YYYY
''
'NEXT.FRAME3'
''
LET FRAME3 = 1
FOR J = 1 TO 12, DO
 IF INFO(CALLING.NODE,J,1) GT 0 OR
    INFO(CALLING.NODE,J,4) GT 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
 ALWAYS
 IF SPECINFO(CALLING.NODE,CALLED.NODE,J) EQ 6 AND
    INFO(CALLED.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,1) EQ 0 AND
    INFO(CALLING.NODE,J,4) EQ 0
  LET SPECINFO(CALLED.NODE,CALLING.NODE,J) = 0
  LET SLOT3 = J
  GO TO CONT1
 ALWAYS
LOOP
''
FOR J = 1 TO 12, DO
 IF INFO(CALLING.NODE,J,1) EQ 0 AND
```

```
    INFO(CALLING.NODE,J,4) EQ 0 AND
    INFO(CALLED.NODE,J,1) EQ 0
 LET SLOT3 = J
 GO TO CONT1
ALWAYS
LOOP
''
'YYYY'
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** FAILED IN EVENT RESPONSE TO REQUEST
SKIP 1 OUTPUT LINE
''
'UNSUCCESS'
''
LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
LET CKT.FAILED = CKT.FAILED + 1
LET BK.TO.DEST = BK.TO.DEST - 1
LET BK.TO.ORG = BK.TO.ORG + 1
''
IF PRNT EQ 0
 PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AND TIME.V
 AS FOLLOWS
 CIRCUIT **** FAILED TO CONNECT AT TIME ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET DIRECTION(MESSAGE) = 3
LET START.TIME(MESSAGE) = TIME.V
''
SCHEDULE A TO.ORG.BREAKDOWN GIVEN MESSAGE
AT TIME.V + BREAKTIME
RETURN
''
''  CONT1 IDENTIFIES A  SLOT TO CARRY THE SERVICE MSG
''  TO THE CALLED NODE AND ALSO COMPUTES WHEN THE
''  SERVICE MESSAGE WILL ARRIVE AT THE CALLED NODE
''
''
'CONT1'
''
 LET DELAY3 = REAL.F(12*FRAME3 + SLOT3 - CURRENT.SLOT)
              * SLOT.DURATION
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT3
LET SLOT.ASSIGN(MESSAGE) = 0
LET RECSLOT(MESSAGE) = 0
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE),FM.NODE(MESSAGE)
              AND (TIME.V + DELAY3) AS FOLLOWS
CKT ***** HAS SCHEDULED A REQ FOR SERVICE AT NODE **
AT TIME ***.*****
SKIP 2 OUTPUT LINES
ALWAYS
''
SCHEDULE A REQUEST.FOR.SVC GIVEN MESSAGE
AT TIME.V + DELAY3
''
IF PRNT EQ 0
PRINT 2 LINES WITH CKT.NUMBER(MESSAGE),FM.NODE(MESSAGE),
         AND (TIME.V + DELAY3)  AS FOLLOWS
 CKT **** HAS SCHEDULED A RESPONSE TO SVC AT NODE **
 AT TIME ***.**
SKIP 1 OUTPUT LINES
''
PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT END OF RESPONSE TO SVC ARE :
```

```
LIST ATTRIBUTES OF MESSAGE
SKIP 2 OUTPUT LINES
ALWAYS
''
RETURN
END  '' RESPONSE TO REQUEST
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT TO.DEST.BREAKDOWN GIVEN BDTODEST
''
''
'' THIS EVENT BREAKS DOWN A ESTABLISHED CIRCUIT
'' FROM THE ORIGINATOR TO THE DESTINATION
''
'' IT REMOVES SLOT ASSIGNMENTS FROM THE NODAL
'' SLOT ASSIGNMENT TABLES SO THAT THESE RELEASED SLOTS
'' CAN BE USED IN THE ESTABLISHMENT OF OTHER CIRCUITS
''
'' THIS EVENT SELECTS A RELEVANT PORTION OF PROGRAM TO
'' EXECUTE DEPENDING ON THE VALUE OF DIRECTION(MESSAGE)
''
''    -2 : START BREAKING DOWN AN ESTABLISHED CIRCUIT
''         FROM THE ORIGINATOR NODE TO THE DESTINATION
''
''    -1 : CONTINUE BREAKING DOWN AN ESTABLISHED CIRCUIT
''         FROM AN INTERMEDIATE NODE TO THE DESTINATION
''
''     0 : BREAK DOWN WHEN A RESPONSE TO REQ FAILED
''
LET MESSAGE = BDTODEST
''
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
TO.DEST BREAK DOWN PERFORMED AT TIME ****.******
 SKIP 2 OUTPUT LINES
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY AT START OF TO.DEST BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
IF TYPE(MESSAGE) EQ 1
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
''
IF DIRECTION(MESSAGE) EQ -1
  GO TO CONT.BREAKDOWN
ALWAYS
''
 IF DIRECTION(MESSAGE) EQ 0
  GO TO RESPONSE.BREAKDOWN
ALWAYS
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ -2 AND
TYPE(MESSAGE) EQ FULL.BREAKDOWN
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       TIME.V AND START.TIME(MESSAGE) AS FOLLOWS
CIRCUIT **** FROM  ** TO  ** WAS ONCE ESTABLISHED
```

```
BROKEN DOWN AT TIME ****.****** AFTER CARRYING VOICE
TRAFFIC FOR A CALL DURATION OF ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
''
LET BK.TO.DEST = BK.TO.DEST + 1
LET DIRECTION(MESSAGE) = -1
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT2.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4)
      = INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET SPECINFO(FM.NODE(MESSAGE),TO.NODE(MESSAGE),I) = 6
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  GO TO COMPUTE.DELAY
 ALWAYS
''
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN TO.DEST BREAKDOWN FOR CIRCUIT NO. *****
SKIP 1 OUTPUT LINE
RETURN
''
''
''  WE HAVE SET THE TRANSMIT AND RECEIVE SLOTS AT THE
''  ORIGINATOR NODE TO ZERO.
''  WE NOW BREAK DOWN THE CIRCUIT ALONG THE UPSTREAM PATH
''
''  CHECK WHETHER WE ARE AT THE DESTINATION NODE ,
''  IF SO ,WE NEED ONLY TO DELETE THE TRANSMIT AND RECEIVE
''  SLOT ASSIGNMENTS FOR THIS CIRCUIT AND
''  COLLECT STATISTICS DATA
''
''
'CONT.BREAKDOWN'
''
  LET SLOT1.XMIT = RECSLOT(MESSAGE)
  LET SLOT1.REC = INFO(TO.NODE(MESSAGE),SLOT1.XMIT,2)
  LET TOMES = TO.NODE(MESSAGE)
  LET INFO(TOMES,SLOT1.XMIT,1) = 0
  LET SPECINFO(TOMES,FM.NODE(MESSAGE),SLOT1.XMIT) = 6
  LET INFO(TO.NODE(MESSAGE),SLOT1.XMIT,2) = 0
  LET INFO(TO.NODE(MESSAGE),SLOT1.XMIT,3) = 0
  LET INFO(TO.NODE(MESSAGE),SLOT1.REC,4)
      = INFO(TO.NODE(MESSAGE),SLOT1.REC,4) - 1
''
''  WE HAVE COMPLETED RELEASING THE DOWN-SIDE RECEIVE
''  AND TRANSMIT SLOT ASSIGNMENTS
''
''  IF WE ARE AT THE DESTINATION NODE,WE CAN NOW COLLECT
''  STATISTIC DATA. OTHERWISE, WE WILL CONTINUE BREAKING
''  DOWN THE UP-SIDE SLOT ASSIGNMENTS
''
''
IF TO.NODE(MESSAGE) EQ DESTINATION(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
 RETURN
```

```
ALWAYS
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT2.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4)
      = INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),TO.NODE(MESSAGE),I)=6
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  LET DIRECTION(MESSAGE) = -1
  GO TO COMPUTE.DELAY
 ALWAYS
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** HAS FAULT IN EVENT TO.DEST.BREAKDOWN
SKIP 1 OUTPUT LINE
RETURN
''
''
'' USES THE ASSIGNED TRANSMIT SLOT TO CARRY THE BD
'' MESSAGE TO THE NEXT NODE UPSTREAM ON THE WAY TO
'' THE DESTINATION NODE.
''
'' CALCULATES WHEN THE BREAK DOWN MESSAGE WILL ARRIVE
'' AT THE NEXT NODE
''
''
'COMPUTE.DELAY'
''
IF SLOT2.XMIT GT (CURRENT.SLOT + 1)
 LET DELAY = SLOT2.XMIT - CURRENT.SLOT
 GO TO NEXT.BREAKDOWN
ALWAYS
''
IF SLOT2.XMIT EQ (CURRENT.SLOT + 1)
 LET DELAY = 13
 GO TO NEXT.BREAKDOWN
ALWAYS
''
IF SLOT2.XMIT LT (CURRENT.SLOT + 1)
 LET DELAY = SLOT2.XMIT - CURRENT.SLOT + 12
 GO TO NEXT.BREAKDOWN
ALWAYS
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN TO.DEST BD DELAY CALCULATION FOR CKT ****
SKIP 1 OUTPUT LINE
RETURN
''
''
'NEXT.BREAKDOWN'
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT2.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
SCHEDULE AN TO.DEST.BREAKDOWN GIVEN MESSAGE
AT TIME.V + INCREMENT
GO TO LAST.TO.DEST
''
'RESPONSE.BREAKDOWN'
''
IF RECSLOT(MESSAGE) EQ 13
```

```
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
ALWAYS
''
IF RECSLOT(MESSAGE) LE 12
 DESTROY THE MESSAGE CALLED MESSAGE
ALWAYS
''
''
''LAST.TO.DEST'
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY AT END OF TO.DEST.BREAKDOWN ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 1 OUTPUT LINE
ALWAYS
''
''
RETURN
END '' TO.DEST.BREAKDOWN
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
EVENT TO.ORG.BREAKDOWN GIVEN BDTOORG
''
''
'' THIS EVENT BREAKS DOWN A ESTABLISHED CIRCUIT
'' FROM THE DESTINATION TO THE SOURCE NODE.
''
'' IT REMOVES SLOT ASSIGNMENTS FROM THE NODAL SLOT
'' ASSIGNMENT TABLES SO THAT THESE RELEASED SLOTS
'' CAN BE USED IN THE ESTABLISHMENT OF OTHER CIRCUITS
''
'' THIS EVENT SELECTS A RELEVANT PORTION OF PROGRAM TO
'' EXECUTE DEPENDING ON THE VALUE OF DIRECTION(MESSAGE)
''
''    1 : START BREAKING DOWN AN ESTABLISHED CIRCUIT FROM
''        THE DESTINATION NODE TO THE SOURCE
''
''    2 : CONTINUE BREAKING DOWN AN ESTABLISHED CIRCUIT
''         FROM AN INTERMEDIATE NODE TO THE SOURCE
''
''    3 : START BREAKING DOWN FROM A NODE TO THE SOURCE
''        CALLED BY REQUEST FOR SERVICE
''
''    4 : START BREAKING DOWN FROM A NODE TO THE ORIGINATOR
''        CALLED BY RESPONSE TO REQUEST
''
''
LET MESSAGE = BDTOORG
''
DEFINE INCREMENT AS A REAL VARIABLE
''
IF PRNT EQ 0
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 TO.ORG.BREAKDOWN PERFORMED AT TIME ****.******
 SKIP 2 OUTPUT LINES
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF THE AT START OF TO.ORG ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
IF TYPE(MESSAGE) EQ 1
```

```
  LET TYPE(MESSAGE) = PARTIAL.BREAKDOWN
ALWAYS
''
''
LET CURRENT.SLOT = SLOT.ARRIVAL(MESSAGE)
''
IF DIRECTION(MESSAGE) EQ  1
  GO TO ONE
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  2
  GO TO TWO
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  3
  GO TO THREE
ALWAYS
''
IF DIRECTION(MESSAGE) EQ  4
  GO TO FOUR
ALWAYS
''
''
''
'ONE'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 1
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
 CKT **** FM ** TO NODE ** WAS ESTABLISHED FOR A CALL
 DURATION OF ****.******* SECS IS BEING  BROKEN DOWN
 AT TIME ****.****** SECS
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET FM.NODE(MESSAGE) = DESTINATION(MESSAGE)
LET START.TIME(MESSAGE) = TIME.V
LET BK.TO.ORG = BK.TO.ORG + 1
LET DIRECTION(MESSAGE) = 2
''
'JUMP.IN'
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT1.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET MM = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = MM
  LET INFO(FM.NODE(MESSAGE),MM,4)
      = INFO(FM.NODE(MESSAGE),MM,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),TO.NODE(MESSAGE),I)=6
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  LET DIRECTION(MESSAGE) = 2
  GO TO COMPUTE.DELAY
 ALWAYS
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
FAULT IN EVENT TO.ORG.BREAKDOWN FOR CIRCUIT NO. *****
SKIP 1 OUTPUT LINE
RETURN
''
''
'' WE HAVE SET THE TRANSMIT AND RECEIVE SLOTS AT THE
'' DESTINATION NODE TO ZERO.
'' WE NOW BREAK DOWN THE CIRCUIT ALONG THE TO.ORG PATH
```

```
''
'' CHECK WHETHER WE ARE AT THE ORIGINATOR NODE, IF SO,
'' WE NEED ONLY TO DELETE THE TRANSMIT AND RECEIVE
'' SLOT ASSIGNMENTS FOR THIS CIRCUIT AND
'' COLLECT STATISTICS DATA
''
''
'TWO'
''
LET SLOT2.XMIT = RECSLOT(MESSAGE)
LET SLOT2.REC = INFO(TO.NODE(MESSAGE),SLOT2.XMIT,2)
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,1) = 0
LET SPECINFO(TO.NODE(MESSAGE),FM.NODE(MESSAGE)
            ,SLOT2.XMIT) = 6
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,2) = 0
LET INFO(TO.NODE(MESSAGE),SLOT2.XMIT,3) = 0
LET INFO(TO.NODE(MESSAGE),SLOT2.REC,4)
    = INFO(TO.NODE(MESSAGE),SLOT2.REC,4) - 1
''
'' WE HAVE COMPLETED RELEASING THE UP-SIDE RECEIVE AND
'' TRANSMIT SLOT ASSIGNMENTS
''
'' IF WE ARE AT THE ORIGINATOR NODE,WE CAN NOW COLLECT
'' STATISTICS. OTHERWISE, WE WILL CONTINUE BREAKING
'' DOWN THE DOWN SIDE SLOT ASSIGNMENTS
''
''
IF TO.NODE(MESSAGE) EQ ORIGINATOR(MESSAGE)
 LET START.TIME(MESSAGE) = TIME.V - START.TIME(MESSAGE)
 PERFORM STATS.AT.END.BREAK.DOWN GIVEN MESSAGE
 RETURN
ALWAYS
''
LET FM.NODE(MESSAGE) = TO.NODE(MESSAGE)
''
FOR I = 1 TO 12 , DO
 IF INFO(FM.NODE(MESSAGE),I,1) EQ CKT.NUMBER(MESSAGE)
  LET SLOT1.XMIT = I
  LET TO.NODE(MESSAGE) = INFO(FM.NODE(MESSAGE),I,3)
  LET M = INFO(FM.NODE(MESSAGE),I,2)
  LET RECSLOT(MESSAGE) = M
  LET INFO(FM.NODE(MESSAGE),M,4)
      = INFO(FM.NODE(MESSAGE),M,4) - 1
  LET INFO(FM.NODE(MESSAGE),I,1) = 0
  LET SPECINFO(FM.NODE(MESSAGE),TO.NODE(MESSAGE),I) = 6
  LET INFO(FM.NODE(MESSAGE),I,2) = 0
  LET INFO(FM.NODE(MESSAGE),I,3) = 0
  LET DIRECTION(MESSAGE) = 2
  GO TO COMPUTE.DELAY
 ALWAYS
''
LOOP
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
CIRCUIT **** HAS FAULT IN EVENT TO.ORG.BEAKDOWN
RETURN
''
''
'THREE'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 3
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
 CKT ***** FROM ** TO NODE ** CANNOT BE ESTABLISHED.
 BEGIN TO BREAK DOWN THE CIRCUIT AT TIME ****.******
 TIME NOW IS ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
```

```
''
LET DIRECTION(MESSAGE) = 2
''
GO TO JUMP.IN
''
'FOUR'
''
IF PRNT EQ 0 AND DIRECTION(MESSAGE) EQ 4
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       START.TIME(MESSAGE) AND TIME.V AS FOLLOWS
 CKT ***** FROM NODE ** TO ** CANNOT BE ESTABLISHED.
 BEGIN TO BREAK DOWN THE CIRCUIT AT TIME ****.******
 TIME NOW IS ****.******
 SKIP 2 OUTPUT LINES
ALWAYS
''
LET DIRECTION(MESSAGE) = 2
''
GO TO JUMP.IN
''
'COMPUTE.DELAY'
''
IF SLOT1.XMIT GT (CURRENT.SLOT + 1)
 LET DELAY = SLOT1.XMIT - CURRENT.SLOT
 GO TO LAST.DOWN
ALWAYS
''
IF SLOT1.XMIT EQ (CURRENT.SLOT + 1)
 LET DELAY = 13
 GO TO LAST.DOWN
ALWAYS
''
IF SLOT1.XMIT LT (CURRENT.SLOT + 1)
 LET DELAY = SLOT1.XMIT - CURRENT.SLOT + 12
 GO TO LAST.DOWN
ALWAYS
''
PRINT 1 LINE WITH CKT.NUMBER(MESSAGE) AS FOLLOWS
 FAULT IN TO.ORG.BD DELAY COMPUTATION AT CKT ****
RETURN
''
'LAST.DOWN'
''
LET SLOT.ARRIVAL(MESSAGE) = SLOT1.XMIT
LET INCREMENT = REAL.F(DELAY) * SLOT.DURATION
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY AT END OF TO.ORG.BD ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 1 OUTPUT LINE
ALWAYS
''
SCHEDULE A TO.ORG.BREAKDOWN GIVEN MESSAGE
AT TIME.V + INCREMENT
''
RETURN
END '' TO.ORG.BREAKDOWN
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR VIRTUAL.CKT GIVEN ESTABLISH.MSG
''
''
'' THIS ROUTE COLLECTS STATISTICS ON CIRCUITS THAT ARE
```

```
'' ESTABLISHED AND SCHEDULES THEIR EVENTUAL
'' DISESTABLISHMENT ACCORDING TO AN EXPONENTIAL
'' DISTRIBUTION FUNCTION WITH A MEAN CALL DURATION
'' OF 20 SECS
''
LET MESSAGE = ESTABLISH.MSG
''
DEFINE CALL.END.TIME AS A REAL VARIABLES
''
IF PRNT EQ 1
 PRINT 1 LINE WITH TIME.V AS FOLLOWS
 ROUTINE VIRTUAL CIRCUIT PERFORMED AT ****.******
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
 ATTRIBUTES OF ENTITY WHEN VIRTUAL CKT WAS CALLED ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
LET CKT.ESTAB = CKT.ESTAB + 1
LET BK.TO.DEST = BK.TO.DEST - 1
LET DELAY.SUM = DELAY.SUM + START.TIME(MESSAGE)
LET AVG.TIME.EST = DELAY.SUM /REAL.F(CKT.ESTAB)
''
''
'' DID THIS CIRCUIT TAKE THE MOST TIME TO ESTABLISH
''
''
IF START.TIME(MESSAGE) GT LONG.TIME.EST
 LET LONG.TIME.EST = START.TIME(MESSAGE)
 LET CKT.LONG.TIME.EST = CKT.NUMBER(MESSAGE)
ALWAYS
''
''
'' SCHEDULES THE TIME FOR THE NEWLY ESTABLISHED CIRCUIT
'' TO BE ACTIVE AND SELECTS FROM EITHER ORIGINATOR NODE
'' OR DESTINATION THE CIRCUIT TO BE DISESTABLISHED AND
'' SCHEDULES THE EVENT TO BREAK DOWN THE CIRCUIT
''
''
LET CALL.DURATION = EXPONENTIAL.F(MEAN.CALL.DURATION,6)
LET CALL.END.TIME = CALL.DURATION + TIME.V
LET SUM.DURATION = SUM.DURATION + CALL.DURATION
LET AVG.DURATION = SUM.DURATION / REAL.F(CKT.ESTAB)
LET START.TIME(MESSAGE) = CALL.DURATION
LET TYPE(MESSAGE) = FULL.BREAKDOWN
''
'' RANDOMLY SELECT A CURRENT SLOT
''
LET SLOT.ARRIVAL(MESSAGE) = RANDI.F(1,12,4)
IF PRNT EQ 0
 PRINT 1 LINE WITH SLOT.ARRIVAL(MESSAGE) AS FOLLOWS
CIRCUIT BEGIN BREAKING DOWN IN SLOT **
 SKIP 1 OUTPUT LINE
ALWAYS
''
''
IF FAIR.POINTER EQ 1
 LET FAIR.POINTER = 0
 LET FM.NODE(MESSAGE) = ORIGINATOR(MESSAGE)
 LET DIRECTION(MESSAGE) = -2
 SCHEDULE AN TO.DEST.BREAKDOWN GIVEN MESSAGE
 AT CALL.END.TIME
 GO TO LAST.VIRTUAL
ALWAYS
```

```
''
IF FAIR.POINTER EQ 0
 LET FAIR.POINTER = 1
 LET DIRECTION(MESSAGE) = 1
 SCHEDULE A TO.ORG.BREAKDOWN GIVEN MESSAGE
 AT CALL.END.TIME
ALWAYS
''
'LAST.VIRTUAL'
''
IF PRNT EQ 0 AND FAIR.POINTER EQ 0
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       TIME.V, CALL.DURATION AND CALL.END.TIME
               AS FOLLOWS
CKT ***** FM NODE ** TO ** WAS ESTABLISHED AT TIME
***.***** AND HAS CALL DURATION OF ****.****** SECS,
 BD WILL BEGIN IN THE TO.DEST DIRECTION AT ***.******
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 0 AND FAIR.POINTER EQ 1
 PRINT 3 LINES WITH CKT.NUMBER(MESSAGE),
       ORIGINATOR(MESSAGE), DESTINATION(MESSAGE),
       TIME.V, CALL.DURATION AND CALL.END.TIME
               AS FOLLOWS
CKT ***** FM ** TO NODE ** WAS ESTABLISHED AT TIME
***.***** AND HAS CALL DURATION OF ****.****** SECS,
BD WILL BEGIN IN THE TO.ORG DIRECTION AT ****.******
 SKIP 1 OUTPUT LINE
ALWAYS
''
IF PRNT EQ 0
 PRINT 1 LINE AS FOLLOWS
ATTRIBUTES OF ENTITY AT END OF VIRTUAL.CKT ARE :
 LIST ATTRIBUTES OF MESSAGE
 SKIP 2 OUTPUT LINES
ALWAYS
''
''
RETURN
END  '' VIRTUAL CKT
''
''
''
''
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR STATS.AT.END.BREAK.DOWN GIVEN B.D.MESSAGE
''
''
'' THIS ROUTINE COLLECTS STATISTICS OF THE CIRCUIT
'' THAT ARE BROKEN DOWN
''
'' IT IS CALLED BY TO.DEST OR TO.ORG BREAKDOWN EVENTS
''
''
LET MESSAGE = B.D.MESSAGE
''
DEFINE BD.TIME AS A REAL VARIABLE
''
IF TYPE(MESSAGE) EQ FULL.BREAKDOWN
 LET CKT.DISESTAB = CKT.DISESTAB + 1
ALWAYS
''
LET CKTS.BD = CKT.DISESTAB + CKT.FAILED
LET BK.TO.ORG = BK.TO.ORG - 1
LET BD.TIME = START.TIME(MESSAGE)
LET SUM.BD.TIME = SUM.BD.TIME + BD.TIME
```

```
LET AVG.BD.TIME = SUM.BD.TIME / REAL.F(CKTS.BD)
''
'' COLLECTS STATS ON THE BREAKDOWN OF PARTIAL
'' ESTABLISHED CIRCUITS
''
IF TYPE(MESSAGE) EQ PARTIAL.BREAKDOWN
 IF START.TIME(MESSAGE) GT LONG.P.BD
  LET LONG.P.BD = START.TIME(MESSAGE)
 ALWAYS
''
 LET TOT.P.BD = TOT.P.BD + START.TIME(MESSAGE)
 LET P.BD.COUNTER = P.BD.COUNTER + 1
 LET AVG.P.BD = TOT.P.BD / REAL.F(P.BD.COUNTER)
ALWAYS
''
''
IF TYPE(MESSAGE) EQ FULL.BREAKDOWN
 IF START.TIME(MESSAGE) GT LONG.C.BD
  LET LONG.C.BD = START.TIME(MESSAGE)
 ALWAYS
''
 LET TOT.C.BD = TOT.C.BD + START.TIME(MESSAGE)
 LET C.BD.COUNTER = C.BD.COUNTER + 1
 LET AVG.C.BD = TOT.C.BD / REAL.F(C.BD.COUNTER)
ALWAYS
''
DESTROY THE MESSAGE CALLED MESSAGE
''
RETURN
''
END '' STATS AT END BREAKDOWN
''
''
'' $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
''
EVENT HALT.SIMULATION SAVING THE EVENT NOTICE
''
''
'' THIS ROUTINES HALTS THE PROGRAM AND PRINTS
'' STATISTICS AND ANALYSIS STATEMENTS
''
''
LET PRNT.COUNTER = PRNT.COUNTER + 1
''
START NEW PAGE
PRINT 1 LINE WITH PRNT.COUNTER AS FOLLOWS
THIS IS THE ** TH SIMULATION RUN
SKIP 1 OUTPUT LINE
''
LET FRACT.SUCCESSFUL.CALL = REAL.F(CKT.ESTAB) * 100.00
                             / REAL.F(CKT.SUM)
LET FRACT.LOST.CALL = 100.0 - FRACT.SUCCESSFUL.CALL
''
PRINT 15 LINES WITH  CKT.SUM, CKT.ESTAB, CKT.DISESTAB,
         CKT.FAILED, OFFERED.TRAFFIC, AVG.TIME.EST,
         LONG.TIME.EST,CKT.LONG.TIME.EST, AVG.DURATION,
         P.BD.COUNTER, C.BD.COUNTER, AVG.C.BD, SLOT.DEPTH,
         FRACT.SUCCESSFUL.CALL AND FRACT.LOST.CALL
                  AS FOLLOWS
STATISTICS OF THIS SIMULATION :
NUMBER OF CIRCUIT CREATED SO FAR = *****
NUMBER OF CIRCUIT ESTABLISHED = ****
NUMBER OF ESTABLISHED CKTS THAT ARE DISESTABLISHED ****
NUMBER OF CIRCUITS WERE NOT ESTABLISHED = ****
OFFERED TRAFFIC IS **
AVERAGE TIME TO ESTABLISH A CKT = ***.******
LONGEST TIME TO ESTABLISH A CKT = ***.****** AT CKT ****
AVERAGE DURATION OF AN ESTABLISHED CIRCUITS = ***.******
NUMBER OF PARTIALLY ESTABLISHED CIRCUITS = ****
```

```
NUMBER OF FULLY ESTABLISHED CIRCUITS = ****
AVERAGE TIME TO BREAK DOWN A COMPLETED CKT = ****.******
SLOT DEPTH IS **
PERCENTAGES OF SUCCESSFULL CALL = ***.*****
PERCENTAGES OF LOST CALL = ***.******
SKIP 3 OUTPUT LINES
''
''
FOR NODE = 1 TO 11, DO
  RESERVE SLOTS.PER.FRAME(*) AS 12
  LET EMPTY = 0
  LET TRANSMIT.SLOTS = 0
  LET RECEIVE.SIGS = 0
''LET REC.SLOTS = 0
''
  FOR S = 1 TO 12 , DO
   IF INFO(NODE,S,4) GE 1
    LET RECEIVE.SIGS = RECEIVE.SIGS + INFO(NODE,S,4)
    LET REC.SLOTS = REC.SLOTS + 1
    LET SLOTS.PER.FRAME(S) = INFO(NODE,S,4)
    GO TO OUT
''  ALWAYS
''
   IF INFO(NODE,S,1) GT 0
    LET TRANSMIT.SLOTS = TRANSMIT.SLOTS + 1
    LET SLOTS.PER.FRAME(S) = 7000 + INFO(NODE,S,3)
    GO TO OUT
''  ALWAYS
''
   IF INFO(NODE,S,1) EQ 0 AND INFO(NODE,S,4) EQ 0
    LET EMPTY = EMPTY + 1
    LET SLOTS.PER.FRAME(S) = 0
''  ALWAYS
''
''OUT'
''
''  LOOP
''
  PRINT 2 LINES WITH NODE, EMPTY, TRANSMIT.SLOTS,
              RECEIVE.SIGS AND REC.SLOTS AS FOLLOWS
   NODE ** HAS ** EMPTY SLOTS, ** TRANSMIT SLOTS, AND
   HAS ** RECEIVE SIGNALS STACKED IN ** RECEIVE SLOTS
''  SKIP 2 OUTPUT LINES
''
''  PRINT THE TIME SLOT ASSIGNMENT AT EACH NODE
''
''
  PRINT 1 LINE WITH SLOTS.PER.FRAME(1),
           SLOTS.PER.FRAME(2),
           SLOTS.PER.FRAME(3),SLOTS.PER.FRAME(4),
           SLOTS.PER.FRAME(5),SLOTS.PER.FRAME(6),
           SLOTS.PER.FRAME(7),SLOTS.PER.FRAME(8),
           SLOTS.PER.FRAME(9),SLOTS.PER.FRAME(10),
           SLOTS.PER.FRAME(11),
           AND SLOTS.PER.FRAME(12)
            AS FOLLOWS
**** **** **** **** **** **** **** **** **** **** **** ****
''SKIP 2 OUTPUT LINES
''
LOOP
''
PERFORM TERMINATION
''
RETURN
END '' HALT.SIMULATION
''
''
''
''
```

```
'' &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
''
ROUTINE FOR TERMINATION
''
FOR EACH NEW.CALL IN EV.S(I.NEW.CALL), DO
 CANCEL THE NEW.CALL
 DESTROY THE NEW.CALL
LOOP
''
FOR EACH REQUEST.FOR.SVC IN EV.S(I.REQUEST.FOR.SVC),DO
 CANCEL THE REQUEST.FOR.SVC
 DESTROY THE REQUEST.FOR.SVC
LOOP
''
FOR EACH RESPONSE.TO.REQUEST
IN EV.S(I.RESPONSE.TO.REQUEST), DO
 CANCEL THE RESPONSE.TO.REQUEST
 DESTROY THE RESPONSE.TO.REQUEST
LOOP
''
FOR EACH TO.DEST.BREAKDOWN
IN EV.S(I.TO.DEST.BREAKDOWN), DO
 CANCEL THE TO.DEST.BREAKDOWN
 DESTROY THE TO.DEST.BREAKDOWN
LOOP
''
FOR EACH TO.ORG.BREAKDOWN
IN EV.S(I.TO.ORG.BREAKDOWN), DO
 CANCEL THE TO.ORG.BREAKDOWN
 DESTROY THE TO.ORG.BREAKDOWN
LOOP
''
FOR EACH YEN.ROUTING IN EV.S(I.YEN.ROUTING), DO
 CANCEL THE YEN.ROUTING
 DESTROY THE YEN.ROUTING
LOOP
''
FOR EACH SLOT.FOR.YEN IN EV.S(I.SLOT.FOR.YEN), DO
 CANCEL THE SLOT.FOR.YEN
 DESTROY THE SLOT.FOR.YEN
LOOP
''
FOR EACH NSLOT.FOR.YEN IN EV.S(I.NSLOT.FOR.YEN), DO
 CANCEL THE NSLOT.FOR.YEN
 DESTROY THE NSLOT.FOR.YEN
LOOP
''
FOR EACH HALT.SIMULATION IN EV.S(I.HALT.SIMULATION), DO
 CANCEL THE HALT.SIMULATION
 DESTROY THE HALT.SIMULATION
LOOP
''
RETURN
END '' TERMINATION
/*
//GO.SYSIN  DD *
0  1  0  1  0  0  0  0  1  0  1
1  0  1  0  1  0  0  0  0  1  0
0  1  0  1  0  1  0  0  0  0  1
1  0  1  0  1  0  1  0  0  0  0
0  1  0  1  0  1  0  1  0  0  0
0  0  1  0  1  0  1  0  1  0  0
0  0  0  1  0  1  0  1  0  1  0
0  0  0  0  1  0  1  0  1  0  1
1  0  0  0  0  1  0  1  0  1  0
0  1  0  0  0  0  1  0  1  0  1
1  0  1  0  0  0  0  1  0  1  0
0  2  11 4  4  9  4  11 9  2  11
1  0  3  3  5  5  10 5  10 10 1
```

```
2   2   0   4   4   6   6   11  6   11  11
1   3   3   0   5   5   7   7   1   7   1
2   2   4   4   0   6   6   8   8   2   8
9   3   3   5   5   0   7   7   9   9   3
4   10  6   4   8   6   0   8   8   10  10
11  5   11  5   5   7   7   0   9   9   11
1   1   6   1   6   6   8   8   0   10  10
11  2   2   7   2   9   7   9   9   0   11
1 10    3   1   8   3   10  8   8   10  0
/*
```

LIST OF REFERENCES

1. Robert, E. K., Steven, A. G., Jerry B. and Ronald, C. K., Advances in Packet Radio Technology, IEEE Trans. comms, Vol.66, No.11, November 1978.

2. Roshan, L. S., Paulo, T. de Sousa and Ashok, D. I., Network Systems - Modeling, Analysis and Design, Van Nostrand Reinhold Company, 1982.

3. Mowafi, O. A. and Kelly, W. J., Integrated Voice/Data Packet Switching Techniques for Future Military Networks, IEEE Trans. Comms., Vol.COM - 28, No. 9, September 1980.

4. Clark, D. D., Pogran, K. T. and Reed, D. P., An Introduction to Local Area Networks, Proc. IEEE, Vol. 66, pp. 1497-1517, November 1978.

5. Pierce, J., How Far Can Data Loops Go ? IEEE Trans. Comms., Vol.COM - 20, pp. 527 - 530, June 1972.

6. Liu, M. T., Distributed Loop Computer Networks, in Advances in Computers, New York: Academic Press, pp. 163 - 221, 1978.

7. Raymond, L. P., Donald, L. S. and Laurence, B. M., Theory of Spread-Spectrum Communications - A Tutorial, IEEE Trans.comms, Vol.COM-30, No. 5, pp 855 - 883, May 1982.

8. Dixon, R. C., Spread Spectrum Systems, New York: John Wiley & Sons, 1976.

9. Ray, H. Pettit, ECM and ECCM Techniques for Digital Communication Systems, Wadsworth, Inc., November 1978.

10. Morris, D. J., Introduction to Communication Command and Control System New York : Pergamon Press, 1977.

11. Gold, R., Optimal Binary Sequences for Spread Spectrum Multiplexing, IEEE Trans. Info. Th., pp. 619 - 621, October 1967.

12. Bruce, E. Briley, Introduction to Telephone Switching, Bell Telephone Laboratories, Inc., 1983.

13. Kleinrock, L., Queueing System Volume I : Theory, John Wiley & Sons, Inc., pp.103 - 107, 1975.

14. Josef, Puzman and Radoslar, Porizek, Communication Control in Computer Networks, John Wiley & Sons, Ltd., 1980.

15. Cantorn, D.G. and Gerla, Optimal Routing in a Packet Switched Computer Network, IEEE Trans.Comput., Vol.C-23, pp. 1026-1069, October,1974.

16. Gallager, R.G., A Minimun Delay Routing Algorithm using Distributed Computation, IEEE Trans.Comm. , Vol.COM-25, pp. 73-85, January,1977.

17. Segall A., The Modeling of Adative Routing in Data Communication Networks, IEEE Trans.Comm. , Vol.COM-25, pp. 85-95, January,1977.

18. Dijkstra, E. W., A Note on Two Problems in Connexion with Graphs, Vol. 1. pp. 269 - 271, Numerische Mathematik, 1959.

19. Naval Postgraduate School Research Report NPS 55-79-015, A Decentralized Algorithm for Finding the Shortest Paths in Defense Communications Networks, by Yen, J. Y., July 1979.

20. Logan, R. R., Application of a Distributed Routing Algorithm to a Packet Swithched Communication Network, Master's thesis, Naval postgrauate School, December 1983.

21. Grange, J. L. and Gien, M., Flow Control in Computer Networks, Elsevier North - Holland, Inc., 1979.

22. Tritcher, W. K., A Time Slot Assignment Algorithm for a TDMA Packet Radio Network, Master's thesis, Naval Postgraduate School, March 1983.

23. Raymond, T. mercer, A Study of Interference in a Tactical Packet Radio Network, Master's thesis, Naval Postgraduate School, June 1982.

24. Barton, R. F., A Primer on Simulation and Gaming New Jersey, Prentice - Hall, Inc., 1970.

25. Reitman, J., Computer Simulation Applications, Discrete - event Simulation for Synthesis and Analysis of Complex Systems, John Wiley & Sons, inc., 1971.

26. Russell, E.C., Simscript II.5 Programming Language Santa Monica, RAND Corporation, 1968.

BIBLIOGRAPHY

Blanc R. P. and Cotton I. W., Computer Networking, IEEE Press, 1976.

Frank, H. and Frisch, I. T., Communication, Transmission, and Transportation Networks, Addison-Wesley, 1971.

Rosner, R. D., Distributed Telecommunications Networks via satellites Packet Swithing, Wadsworth, 1982.

Schwartz, M., Computer-Communication Network Design and Analysis, Prentice-Hall, 1977.

Tannenbaum, A. S., Computer Networks, Prentice-Hall, 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Prof. J. M. Wozencraft, Code 62Wn Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	2
5.	Prof. J. F. Chang, Code 62Cn Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
6.	Mr. Seah Moon Ming BLK. 1F, Gillman Heights #10-51 Republic of Singapore	3
7.	CPT. Larry Ang BLK.5, Normanton Park, #11-107 Singapore 0511 Republic of Singapore	1
8.	COL. Alaa Fahmy SMC 1716, Naval Postgraduate School Monterey, California 93943	1
9.	LT. Ham, Byung Woon 602-00 Chinhae Naval Academy Faculty E.E Republic of Korea	1
10.	LTC. Albassiouni Abdel SMC 1689, Naval Postgraduate School Monterey, California 93943	1
11.	MAJ. Cho, Joo Hyung SMC 1710, Naval Postgraduate School Monterey, California 93943	1

12. Mr. Serdar Akinsel 1
SMC 1327, Naval Postgraduate School
Monterey, California 93943

Thesis
S40522 Seah
c.1 Voice communications
over packet radio net-
works.

thesS40522
Voice communications over packet radio n
3 2768 000 61240 2
DUDLEY KNOX LIBRARY